UiO : Department of Informatics
University of Oslo

Network Intrusion Detection in Cloud Environments

Ole Jørgen M. Hagen

Master's Thesis Spring 2014

Network Intrusion Detection in Cloud Environments

Ole Jørgen M. Hagen

20th May 2014

Abstract

This master thesis demonstrates an approach for evaluating and ranking different implementations of network intrusion detection systems in cloud environments. The paper will attempt to find what the relevant aspects of different implementations is, and how to structure this information in a coherent way so they can be scored and compared. By focusing on the evaluation process when it comes to protecting cloud environments, the goal is to produce a tool that will give a use case specific recommendation of where to place the network intrusion detection system.

Contents

List of Figures

List of Tables

Acknowledgements

First and foremost I would like to thank Kyrre Begnum for his steady guidance through this master thesis. His knowledge and enthusiasm helped form this thesis and I am forever grateful for his guidance. My gratitude also extends to the rest of the faculty of Oslo Community College, especially Hårek Haugerud, for making this master's programme possible.

A big thanks also goes to family, friends, and fellow classmates. Their help and support really motivated me through this period.

Chapter 1

Introduction

1.1 Motivation

Cloud computing represents a major industry today. In 2012 global spending on cloud services was $110.3B and is expected to grow to $210B in 2016.[7] It will then represent the bulk of new IT spending.[20] The economic gain of moving services to the cloud is an important selling point for businesses and organizations, and is a contributing factor for the sharp increase in usage.

1.1.1 Cloud Computing

Services are valuable to organizations. It is revenue from these services they make their income. Any interruption in their ability to offer a service usually result significant loss of revenue. In cloud environments these are often web services. Their deployment can often be complicated and not easily migrated to other infrastructures. Cloud computing has the benefit of rapid scaling of available resources to meet changes in traffic. This ability to dynamically provision resources is what makes cloud computing valuable from a cost perspective. Where organizations before had to provision hardware to account for maximum load on their service, dynamic scaling allows customers of cloud providers to not pay for more resources than they need at any given time.

The dynamic provisioning of resources in cloud environments is usually in the form of virtual machines being added or removed. This makes the life expectancy of each virtual machine fairly low. Less time is then dedicated to securing each virtual machine, since such security measures often requires spending time installing and configuring software. If the virtual machines are not adequately protected, the security of the service is reduced.

The state of the art when handling security in cloud environments is through security groups. Security groups are sets of IP filter rules that are applied to an instance's networking. This is close to how a firewall

operates and provides roughly the same level of protection. Properly configured security groups limits what kind of traffic that can reach the virtual machines. However traffic on ports needed for the service to operate still need to pass through, and there is no guarantee that some of this traffic is malicious and part of an attack intended to compromise the service. Security groups can only allow or deny traffic based on type of IP protocol and target port. There is no inspection of the traffic itself or the pattern the attack is arriving at. This is a security concern since network attacks often disguises themselves as legitimate traffic intended for open ports on the system.

1.1.2 Network Intrusion Detection System

A Network Intrusion Detection System (NIDS) provides protection in areas a firewall does not cover. It analyses the network traffic for signs of malicious activity. In cloud environments this is traffic already permitted by the security group. A combination of a firewall and intrusion detection software form the basis for a Intrusion Prevention System (IPS). The firewall is here told to drop packets flagged as malicious by the IPS so it never reaches its destination.

Network Intrusion Detection (NID) is currently not implemented as a service in popular cloud services such as Amazon EC2[1] or Google Compute Engine.[10] Nor can it be found on the current road map for developers. This is for various reasons. Clouds are large and complex. There is an enormous amount of network traffic in data centers hosting cloud services. Both internally within the data center and externally to the rest of the Internet. Implementing NID to cover partial or the whole amount of network traffic means reserving a substantial amount of resources for that task. There is also a development and maintenance cost to consider. With a strict competition on pricing, cloud providers are hesitant to go this route and offer it for free. To offer it as a service with a fee is another option, but no major cloud providers has chosen to implement this. There is also privacy issues regarding having NID inherent in cloud environments. NIDS usually generate alerts when malicious activity is found. These alerts may contain private information the user does not want anyone else to see.

Users of cloud environments have the option to implement NID themselves. Virtual machines running in the cloud can host network intrusion detection software to protect itself or act as a gateway protecting other machines. This increases the complexity of the setup and may conflict with other offered services, such as load balancing. Should the user choose to implement NID on most or all the running virtual machines, it will drive the prices up since larger VMs are needed with more memory and processing power. There is also a performance hit when NIDS is installed on VMs, so more virtual machines may also be needed. This is not attractive from a customer point of view.

1.2 Problem statement

Q1 – How can different implementations of network intrusion detection systems in cloud environments be evaluated and ranked for specific use cases?

Different implementations refers to different locations in the cloud infrastructure. There are several possible locations in the infrastructure that NIDS can operate and capture traffic. Only one NIDS will be used in the different implementations.

To determine which implementation of NIDS in a cloud infrastructure fits a use case best, the different implementations need to be evaluated. Evaluation refers to give a score for the different characteristics an implementation has. This can be positive characteristics such as how much of the network traffic it is able to capture. Or negative characteristics such as how much it impacts the performance of the cloud. Once these characteristics has been scored the different implementations can be ranked against each other to see which fit a specific use case best.

Use cases is cloud providers that need to decide where in their infrastructure they should place their network intrusion detection systems. Two or more alternatives has been found and they need to decide which one best fits their priorities.

Chapter 2

Background

This chapter will briefly cover the concept of a network intrusion detection system and cloud computing. Two technologies that have been chosen for this project related to each concept will be described in more detail.

2.1 Network Intrusion Detection System

A network intrusion detection system is responsible for monitoring network traffic on a packet level. Attacks are found by capturing network packets and analysing them. One or several sensors which are placed in a network for detecting malicious activities can be classified as a network intrusion detection system. There are two approaches that enable NIDS to detect attacks. The first is anomaly detection and the second is signature based detection. Most NIDS use one of these methods, or a combination of them.

Anomaly detection, also known as behaviour-based detection, is a network intrusion detection system which models the normal behaviour of the network. Should a deviation from this normal behaviour be found, an alarm is raised. This enables the NIDS to detect new and unknown attacks, but comes at the expense of more false positive alarms.

Signature based detection, sometimes called misuse detection, can be used for detecting known attacks. It has a higher level of security than anomaly detection, but the problem with signature based NIDS is that every captured packet needs to be compared to signatures for known attacks. This process is time-consuming and slows down the throughput of the NIDS. The majority of network intrusion detection systems use signatures to detect attacks.

2.1.1 Snort

Snort[22] is an open source network intrusion detection and intrusion prevention system. Originally created by Martin Roesch in 1998, it is

not developed by Sourcefire.[21] Snort is capable of performing real-time traffic analysis and packet logging on IP networks. Whether traffic should be stopped and allowed to pass is described by a rule language. New rules and updates to existing rules is being developed by the Sourcefire Vulnerability Research Team (VRT). There are two main releases of these rules. Subscriber Release is for registered users that pay an annual fee and offer immediate access to the most up-to-date rules. Registered User Release is for users that do not want to pay an annual fee, so they get access to the same rules, but 30 days after their initial release.

Modes of operation

Snort can be used as a defence for a variety of attacks and probes, notably CGI attacks, SMB probes and OS fingerprinting attempts. Its detection engine uses a modular plug-in architecture, so it is highly customisable. There are three main modes Snort can be configured in: Sniffer, packet logger, and network intrusion detection. In Sniffer Mode, Snort will read network packets, and depending on configuration, print different information to the screen. This can be TCP/UDP/IMCP headers, packet data, or a combination of those two. In Packet Logger Mode, packets are recorded to the disk. Based upon the IP address of one of the hosts in the datagram, every packet is placed in a directory hierarchy. In intrusion detection mode, Snort will monitor the network traffic. Should traffic match a defined rule set, Snort will perform an action based on which rule is triggered.

Configuration of Snort

The configuration of Snort is specified within the snort.conf file. This file also allows other snort config files to be included within it with the include keyword. Three types of variables may be defined in Snorts configuration file:

- var
- portvar
- ipvar

Var refers to a rule path for the different rules. Portvar is which ports is to be included in the variable. Ipvar refers to which IP addresses or subnets is included. An example of this is seen below:

```
# If you are using reputation preprocessor set these
var WHITE_LIST_PATH /usr/local/snort/rules
var BLACK_LIST_PATH /usr/local/snort/rules

# List of file data ports for file inspection
portvar FILE_DATA_PORTS [$HTTP_PORTS,110,143]
```

```
# List of ssh servers on your network
ipvar SSH_SERVERS $HOME_NET
```

Snort rules

Snort uses a simple, lightweight rules description language.This enables most Snort rules to be written in a single line. Snort rues are divided into two logical sections, the rule header and the rule options. The rule header defines the who, where, and what of a packet, in addition to what to do if a packet matches the signature. An example rule is shown below:

```
alert tcp any any -> 192.168.1.0/24 111 \
   (content:"|00 01 86 a5|"; msg:"mountd access";)
```

2.2 Cloud computing

Cloud computing is a set of resources and services offered through the Internet. Throughout the world data centers are set up by different companies to provide cloud services. It is considered the next generation computing platform that can provide dynamic resource pools, virtualization and high availability.

A cloud can be either private, public, community, or hybrid. A private cloud is used by a single organization. It can be internally or externally hosted. A public cloud is provisioned for open use for the public by a particular organization, who also hosts the service. Community clouds are shared by several organizations and are typically externally hosted, but can be internally hosted by one of the organizations. A hybrid cloud is a composition of two or more clouds (private, community or public) that remain unique entities but are bound together offering the benefits of multiple deployment models.

Companies offering cloud services often choose to build their own proprietary solution. This is the case for Google and its Compute Engine cloud service. Amazon has also built their own solution for their Elastic Compute Cloud. But free and open source solutions also exist for companies that want to build their own private cloud or want to offer a cloud solution to customers. The most notable of these projects, and the one chosen for this project is OpenStack.

2.2.1 OpenStack

OpenStack[19] stems from a group of projects within NASA. When facing the issue of how to host and process high-res images of the Moon and Mars they decided to build their own cloud computing fabric controller. They

wanted it to mimic the functionality offered by Amazon EC2. The project was named Nova and is a way of provisioning on demand virtual resources. At the same time the company Rackspace was building Swift, a redundant scalable storage system. After discovering they had a very similar technical approach to the same goal, which was to build "infrastructure clouds", they decided to combine the projects and in 2010 released OpenStack. OpenStack is a free and open source cloud-computing platform for private and public clouds released under Apache License 2.0. This means everyone is free to use it, and to fork the code if they want.

The release proved a success and the community of developers grew rapidly the following years. Today over 200 companies has joined the project, including big names such as Cisco, Dell, and IBM. The components has grown from 2 to 9 and has completed its goal to act as an Infrastructure as a Service.

List of components:

- **Compute (Nova) -** is a cloud computing fabric controller, the main part of an IaaS. It manages and automate pools of computer resources and can work with several virtualization technologies. Its architecture allows it to scale horizontally on standard hardware.
- **Object storage (Swift) -** is a scalable redundant storage system. Software logic ensures integrity of data by replicating and distributing it across different devices.
- **Block Storage (Cinder) -** is the block storage system that provides persistent block-level storage to compute instances.
- **Networking (Neutron) -** manages networks and IP addresses.
- **Dashboard (Horizon) -** is a graphical interface to access, provision and automate cloud-based resources.
- **Identity Service (Keystone) -** is a central directory of users and acts as a common authentication system.
- **Image Service (Glance) -** is used for discovery, registration and delivery of services for disk and server images.
- **Telemetry (Ceilometer) -** provide counters for billing customers. The counters are auditable and and traceable.
- **Orchestration (Heat) -** is a service to orchestrate multiple composite cloud applications using templates.

Networking in OpenStack

The logical layout for networking in OpenStack is shown in Figure 2.1. When traffic is coming from the Internet to the cloud, it first arrives at the Network node. As seen in the diagram the Network node has three main elements regarding the management of networking. DHCP is a standardized networking protocol used on Internet Protocol (IP). Its function is to dynamically distribute network configuration parameters such as IP addresses. This is done automatically, so there is no need for a network administrator to configure these settings manualy. The DHCP agent allocates IP addresses to the VMs on data networks.

The L2 and L3 agent enables distributed virtual router functionality. This is in reference to the second and third layer of the OSI model.[8] The L2 agent is an Open vSwitch (OVS) agent that enables distributed virtual router functionality at the L2 layer. The L3 agent, known as the "neutron-l3-agent", creates virtual routers that connect L2 networks. It uses the Linux IP stack and iptables to perform L3 forwarding and NAT. Connecting the Network node to the Compute nodes is a physical switch.

Each Compute node can consist of multiple tenants. Tenant, earlier called "project", is basically consumers or customers of the cloud. They are isolated resource containers forming the organzational structure within the Compute servers. Each tenant consist of a separate VLAN, volumes, instance, images keys and users. A user of the cloud can specify which tenant he or she wishes to be known as, or Compute can use a tenant with the same ID as the user. As seen in Figure 2.1 each tenant has an assigned L2 agent on the Compute node.

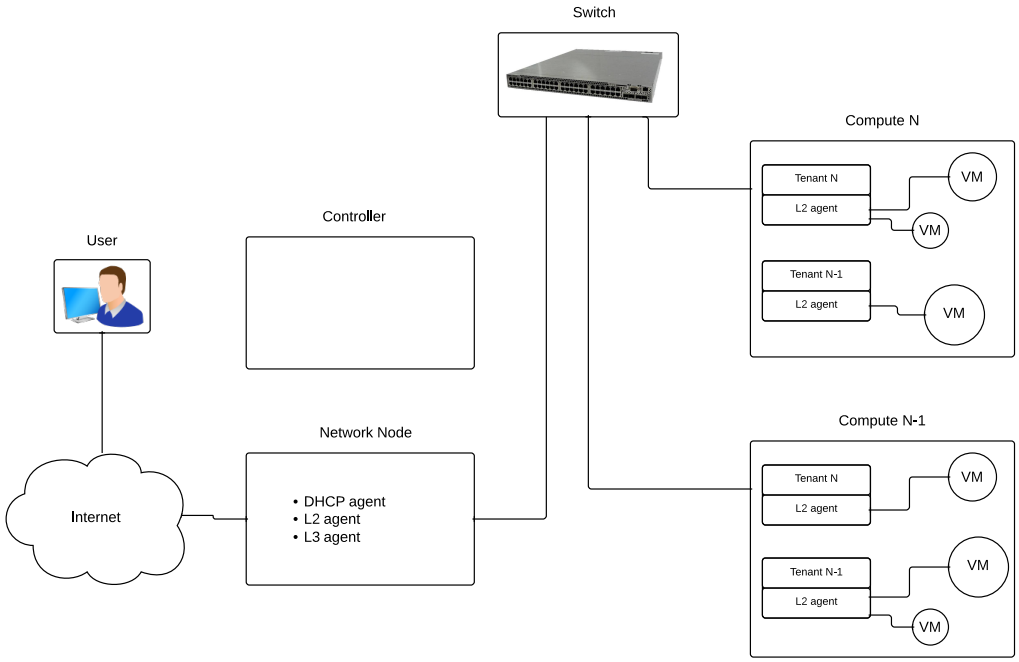

Figure 2.1: Logical network layout in OpenStack.

2.2.2 Other cloud environments

OpenNebula

OpenNebula[18] is a cloud-computing project for managing heterogeneous distributed data center infrastructures. First established as a research project in 2005, it saw its first public release in March 2008. OpenNebula has evolved through open-source releases and now operates as an open source project. This has given it the benefit of an active and engaged community of users and developers. Storage, network, virtualization and security technologies are all supported by OpenNebula.

Eucalyptus

Eucalyptus[9] is an open-source cloud environment for building private and hybrid cloud environments. The software was originally developed at the Virtual Grid Application Development Software project at Rice University, but further development is now handled by Eucalyptus Systems, Inc. Eucalyptus is compatible with AWS and will continue to be so through a formal agreement with AWS. Both Amazon and Eucalyptus instances can be managed with Eucalyptus commands, and instances can

be moved between them. Eucalyptus is written in Java and C, and runs on GNU/Linux operating systems.

Amazon Elastic Compute Cloud

Amazon Elastic Compute Cloud (EC2)[1] saw its first public beta August 2006. It has since become a major player in the cloud computing market. Running on proprietary software developed by Amazon, it is now a core part of Amazon Web Services (AWS).[3] EC2 provides resizeable computing capacity in the EC2 cloud, hence the name elastic. Building on the popularity by the customers from early on, EC2 has all the major functionality expected of a public cloud. Their pricing model has also set a standard for other cloud providers.

2.3 Research in network intrusion detection and cloud environments

As cloud computing becomes more and more popular so does research in cloud environments. Security has remained a constant issue for Open Systems and the Internet. Without security functionality beyond what a firewall can offer inherent in most cloud environments, research into how best to protect the hosted services is an area of ongoing research.

2.3.1 Bayesian Classifier and Snort based Network Intrusion Detection System in Cloud Computing

Modi et al[16] proposed a framework using Bayesian classifier and Snort based network intrusion detection in cloud computing. Its aim is to detect network intrusions in Cloud environments with low false positives and affordable computational cost. The NIDS module was evaluated on KDD'99 experimental dataset. The chosen cloud environment was Eucalyptus, an open source Cloud, installed on Ubuntu operating system.

The NIDS module is installed on each Node Controller (NC) at the back end and all types of traffic is allowed by opening all the ports on Eucalyptus. Scrapy is used for sending custom packets on the network. The NIDS module is capable of detecting a higher number of intrusions with low false positives in the Eucalyptus environment. The proposed NIDS module is also shown to be computationally affordable in addition to being scalable. Each NIDS module can be added or removed at each NC since the modules operate independently.

This work shows the benefits of using signature and anomaly based detection techniques, which is complementing each other. Both known and unknown attacks can then be detected in Cloud. Performance results show

a high detection rate with low false positives, low false negatives and a affordable computational cost.

2.3.2 SnortFlow: A OpenFlow-based Intrusion Prevention System in Cloud Environments

A paper written by Xing et al[24] investigates an OpenFlow and Snort based IPS called "SnortFlow". The cloud environment is based on XenServer[5] that is an efficient parallel virtualization solution. The aim is to demonstrate the feasibility of SnortFlow to detect intrusions and deploy countermeasures by reconfiguring the cloud networking system on-the-fly. OpenFlow is a technology that introduces a centralized and separate controller to enable network programmability. Intrusion detection capability is provided Snort.

Network reconfiguration actions is not always an advisable choice, as stated in the paper. It may kill healthy and innocent traffic. This can affect Service Level Agreements (SLAs). It can also be costly in terms of resources and operational complexity. But increased prevention of different types of attacks makes it an area of promising research.

2.3.3 A Neural Network Based Distributed Intrusion Detection System On Cloud Platform

Sun et al[12] proposed a neural network based distributed intrusion detection system. One of the challenges facing cloud computing is load distribution. This is an important factor when deploying an Intrusion Detection System (IDS). Due to detection overhead, parts of the cloud may be overloaded. A neural network based IDS will distribute the load so that no single machine is overloaded. It also enables machine learning so that new types of attacks can be detected.

According to the results, the accuracy of the implemented IDS is shown to be high, and the time expense is acceptable. This makes neural networking in cloud computing a promising direction of research. The authors noted that much room is left for improving the current work. The current dataset used, KDD, is based on every message passing through a single machine on the network. The algorithm can be enhanced to detect attacks that compromises several machines simultaneously.

2.3.4 Summary

Much of the research in network intrusion detection and cloud environments revolve around improving current models of finding new ones. The research by Modi et al in section 2.3.1 shows that NIDS with low false positive and low false negative can be deployed with an affordable computational cost. Should Snort be the NIDS chosen for implementation in a

cloud infrastructure, a combination of signature and anomaly based detection will give suitable protection against attacks.

New models for protection against network-based attacks is also under research. Most intrusion prevention systems drops packets flagged as malicious by the IPS. Some also choose to block the attacking IP-address, though this offer limited protection. Reconfiguring the cloud networking system when an attack is detected is a new countermeasure investigated by Sun et al (section 2.3.2). This is a feature not offered by traditional IDS/IPS systems.

Techniques for lowering the performance impact a NIDS have on servers in a cloud was seen in many of the relevant articles. Clouds are centralised units hosting services, and generate a large amount of network traffic. An area of research here is therefore to find techniques for lowering the performance impact each NIDS imposes and finding new ways to distribute the load throughout the system. Sun et al proposed a model based on the concept of neural networks (section 2.3.3) that showed how effective load distribution can be done.

While new models for protecting cloud networks are being researched, methods for evaluating different implementations was not found. Cloud providers often have to choose between a set of different security implementations when designing their cloud. Each one usually have their own priorities that will affect the outcome.

2.4 NID access in cloud environments

Network intrusion detection is not implemented in the services of the major cloud providers, but options exist where developers can rent out their own custom images through subscription models. Software installed on these images can provide functionality for network intrusion detection. Below is a review of selected security products sold through AWS Marketplace.[4].

2.4.1 Amazon Web Services Marketplace

Amazon DevPay[2] is a billing and account management service that businesses can use to sell applications that are built in, or run on top of, Amazon Web Services. Services can be sold through Amazon EC2 Machine Images (AMIs), or desktop and web applications that use Amazon S3. DevPay provides a web interface for pricing applications based on any combination of one-time charges, recurring monthly charge, or metered Amazon web services usage charges. Some of these services provide functionality for network intrusion detection along with other security features. Below is a list of security products sold through Amazon Marketplace that provide NID protection.

Alert Logic Threat Manager for AWS (EC2)

Alert Logic Threat Manager for AWS (EC2)[13] provides intrusion detection service (IDS) for Amazon Web Services (AWS) through a 64-bit Amazon Machine Image. Highlighted features are Network Security, Privacy and Control, and Management Portal and API. Network traffic are mirrored from Windows and Linux instances via a soft-tap agent to the virtual appliance.

MetaFlows Security System for EC2

MetaFlows Security System for EC2[11] provides malware detection, intrusion prevention and IT compliance through a 64-bit AMI. Malware detection is supported by geo-location honeypot intelligence, BotHunter and Snort signature analysis. Intrusion prevention comes with advanced analysis and reporting tools. HIPAA, SOX, PCI DSS and other mandates is supported for IT compliance.

CloudPassage Halo

CloudPassage Halo[6], delivered through Amazon EC2 as Software as a Service (SaaS), can be embedded in AMI or deployed through configuration management tools such as Chef or Puppet. Highlighted features is advanced network access control, two-factor authentication, intrusion detection and configuration security. An elastic compute grid perform security analysis on behalf of the virtual servers.

2.4.2 Summary

Security products sold through AWS Marketplace offer NID/IPS functionality to detect and stop attacks. This is beneficial for customers of Amazon EC2 wanting to increase the security of their cloud solution. Having the security provided by an AMI hosted on a separate instance requires the rest of the service to be tailored for it. This can be straightforward or end up being a complex task. It also adds an extra cost to the service.

Chapter 3

Approach

The problem statement asks how different implementations of network intrusion detection systems in cloud environments can be evaluated and ranked for specific use cases. This calls for a study with explorative and comparative elements where prototypes will be created and compared in a cloud environment. The prototypes will not be implemented in the infrastructure, but used as a basis to test the evaluation framework. A framework will be developed to evaluate and rank the different prototypes for different use cases. The chosen network intrusion detection system in this study is Snort. It is widely used and highly acclaimed, and since it is free and open source software, no licence is required to use it. OpenStack is the cloud environment Snort will be implemented in. Released under Apache License 2.0 there is no restrictions on its usage. Its popularity and active development makes it a relevant cloud environment to study.

3.1 Phase I: A closer look at networking in OpenStack to find possible Snort placements

The documentation provided for OpenStack is not sufficient to find viable placements for Snort. The networking therefore need to be explored in more detail, especially how Neutron works. The OpenStack installation used is at Oslo Community College of Applied Sciences[17], called Alto. There are two ways to visualize OpenStacks networking. One is to map the virtual network interfaces enabling the network flow. These are important since these are the interfaces Snort need to listen on to capture traffic. It will also give a view of different checkpoints network traffic pass as they traverse the infrastructure. The other way is to map the logical network layout in OpenStack. This shows the components managing the network at different parts of the infrastructure. Both will be done in this project. Once the networking in OpenStack has been mapped, possible placements for Snort can be decided. 3 or 4 locations will be picked at different parts of OpenStacks infrastructure.

3.2 Phase II: A framework for evaluating Snort placements

When evaluating how to protect a cloud, several factors need to be taken into account. Some may prioritize security over all other factors, if the cloud is housing very sensitive data. Others may have cost concerns, and does not want to allocate too much resources for protection. For someone running performance sensitive services, it is important that the protection network intrusion detection offer does not come at too much of a performance penalty.

A framework will be developed to evaluate different placements of Snort in OpenStack. Its function is to show the strengths and weaknesses of each implementation. Scores will be given based on several factors such as the range of traffic captured, the complexity of the setup, and the performance influence the solution has on the rest of the cloud. The framework will take the placements of Snort as input, and give scores on the different features as output.

3.3 Phase III: Analyse Snort placements using the evaluation framework

Once viable placements for Snort in the OpenStack infrastructure has been found, they can be evaluated using the evaluation framework. This will test the frameworks usefulness in highlighting the strength and weaknesses of each implementation. The framework will enable different alternatives, or combination of alternatives, to be ranked against each other. Different cloud providers will have different priorities when it comes to the properties of the alternative they choose. The evaluation framework need to reflect that. The scores will therefore be weighted so the users of the framework can choose how much each property influences the final sum.

3.4 Experiments

Since OpenStack offer limited documentation when it comes to networking, it is not clear what types of traffic Snort is able to listen on when placed at different parts in the infrastructure. This need to be clear since the evaluation framework will give scores based on the range of traffic captured. This calls for an experiment where Snort is placed at different parts of OpenStacks infrastructure to see what type traffic passes through them. A closer look at the performance characteristics of Snort will also be done. This will highlight if there is unknown features for Snort that should be included in the evaluation framework.

3.4.1 Experiment 1: Test range of traffic captured by Snort at the Network node and Compute nodes

In this experiment the range of traffic captured by Snort at two relevant locations will be tested. The first location is the Network node. The Second location is at a Compute node. Three types of traffic will be sent to a virtual machine at a Compute node. This is traffic that will trigger a custom rule inserted into Snort at both locations. The first traffic is external traffic between the virtual machine and an external computer. The second is internal traffic between the Compute node with Snort installed and another Compute node. The third is local traffic between two virtual machines at the same Compute node. This experiment will show the difference in Snorts ability to capture different ranges of traffic when residing on the Network node and on a Compute node. The evidence that the given traffic has passed a location is that an alert will be found in Snorts alert log.

3.4.2 Experiment 2: Measure performance impact imposed by Snort installations

This experiment will measure the performance impact a Snort installation have on server performance at different rates of traffic. CPU and memory usage will be recorded throughout the benchmark. Snort will be located at the Network node and traffic will be sent from the Internet to a virtual machine running on a Compute node. Both CPU and memory usage are expected to rise when the traffic increases.

Chapter 4

OpenStack networking in detail and possible Snort placements

To find viable placements for Snort in OpenStacks infrastructure, the network layout needed to be explored in more detail. The information is gathered from reading documentation supplied for OpenStack on their homepage.[19] Where documentation fell short a manual search has been done by logging in to HiOAs[17] Alto cloud, which is using OpenStack. From there, the different parts of the infrastructure could be explored and mapped. The flow of the different types of traffic was found by listening on different interfaces with tcpdump[23] and then sending ping requests between elements of OpenStack.

4.1 Network layout in OpenStack

The layout for networking in OpenStack is shown in Figure 4.1. Network traffic coming from the Internet and to the Network node passes through four network interfaces before being sent to the rest of the cloud. The first is eth-ex, the second is br-ex, the third is br-eth-int, and the fourth is eth-int. From there the traffic passes through a physical switch before being sent to the Compute nodes. On a Compute nodes two bridges, br-int and br-eth, is connecting the virtual machines to the eth interface.

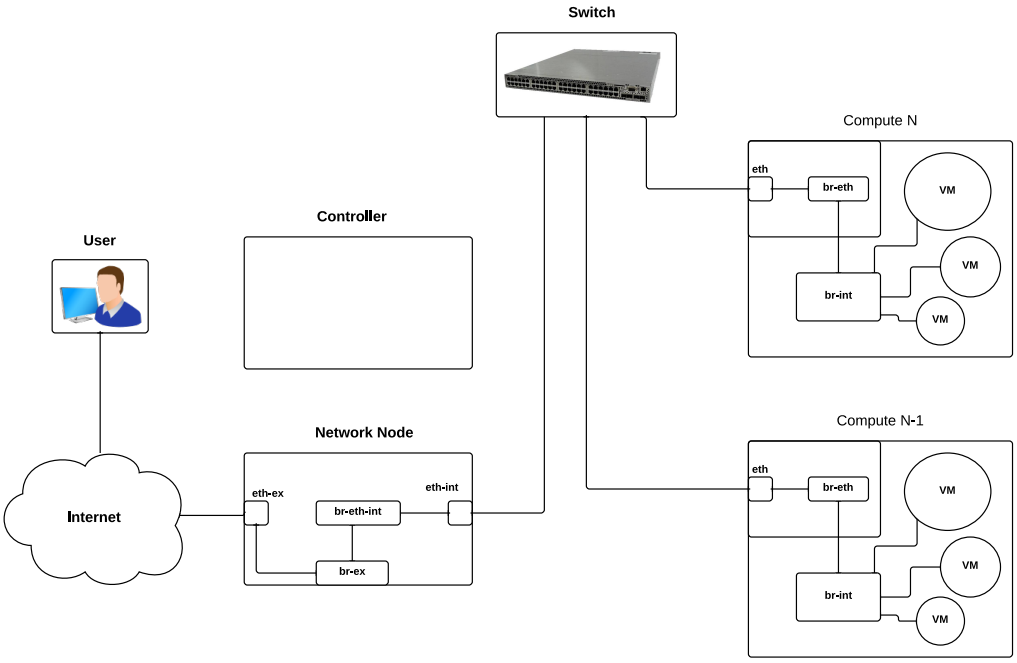

Figure 4.1: Network layout in OpenStack.

4.2 Network flow in OpenStack

There types of traffic in OpenStack is defined: External, internal and local traffic. This is not definitions on traffic that is used by OpenStack in its documentation, but terms assigned in this project.

4.2.1 External traffic

External traffic is traffic between virtual machines running on Compute nodes and the Internet, as shown in Figure 4.2. This traffic will pass through the Network node and the physical switch before arriving at a Compute node.

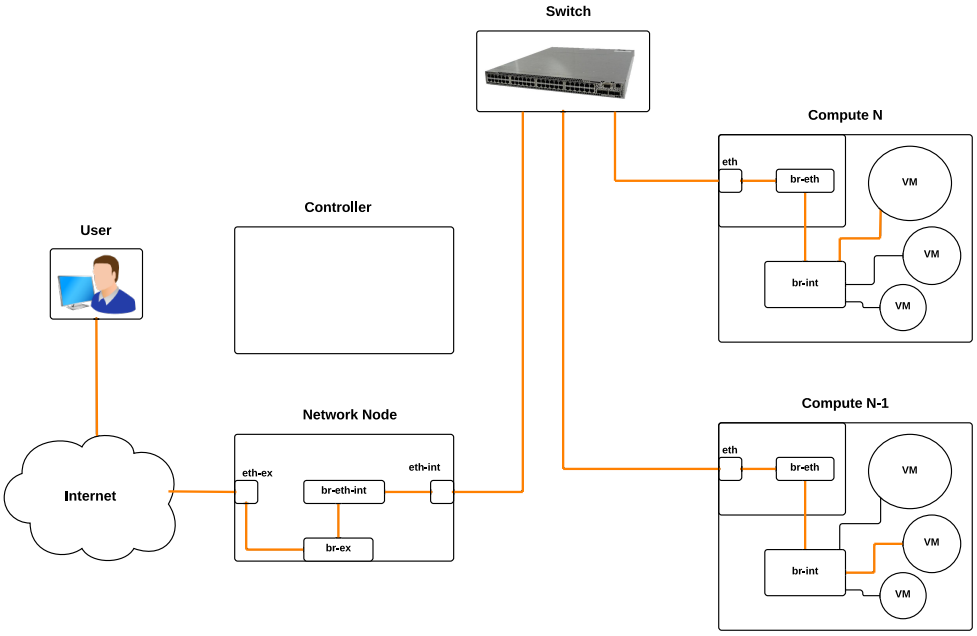

Figure 4.2: Diagram showing the flow of external traffic in OpenStack.

4.2.2 Internal traffic

Internal traffic is traffic between virtual machines running on different Compute nodes, as seen in Figure 4.3. This traffic never reaches the Network node, but is being sent from a Compute node to the physical switch, and then directly to another Compute node.

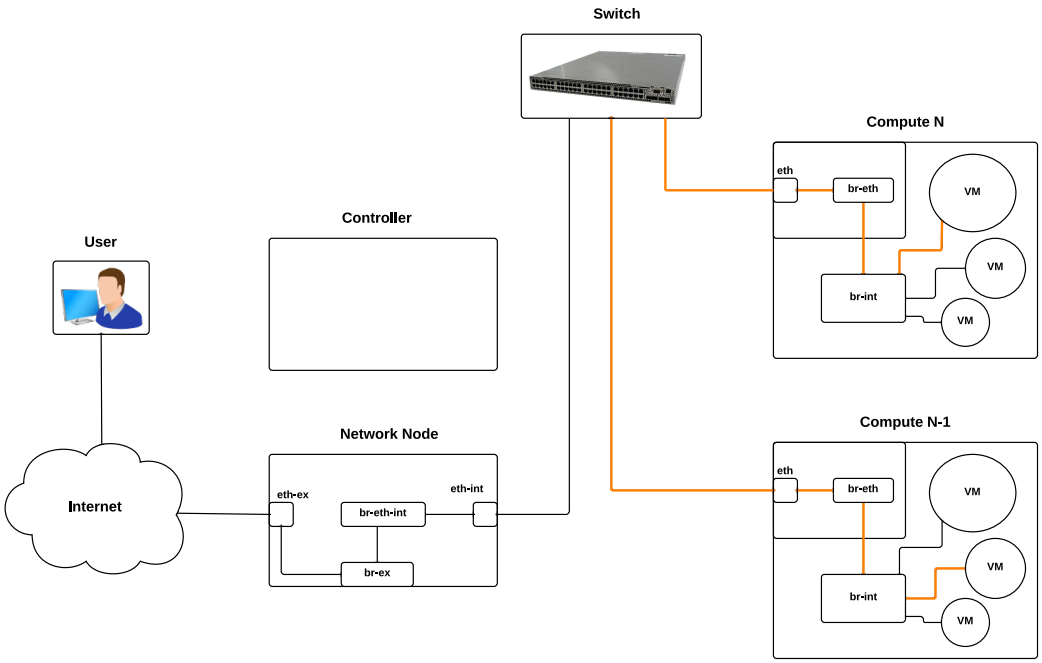

Figure 4.3: Diagram showing the flow of internal traffic in OpenStack.

4.2.3 Local traffic

Local traffic is traffic between virtual machines on the same Compute node. This traffic never leaves the Compute node. It is being sent from a virtual machine to another through the interface br-int, as seen in Figure 4.4.

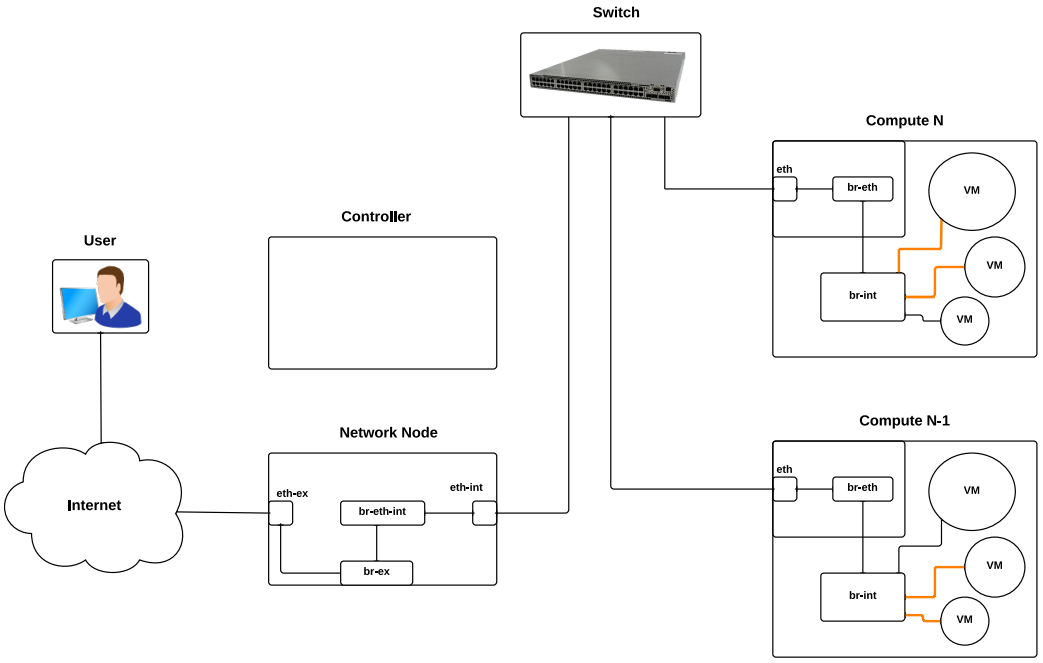

Figure 4.4: Diagram showing the flow of local traffic in OpenStack.

4.3 Possible Snort placements

With the networking of OpenStack mapped several locations stand out as good places to put Snort. The different locations should together cover the whole range of traffic: external, internal, and local. The first location is the Network node. The Network node acts as a gateway between the Internet and the rest of the cloud. Any external network attack on the cloud infrastructure will pass through the Network node. This placement covers external traffic, but not internal or local.

The physical switch is another good location to listen on traffic, since it covers all external and internal traffic. It is not possible to install Snort on the switch itself, but traffic can be mirrored with a span port to a separate machine with Snort installed. This has the advantage that the processing power used by Snort comes from a separate machine, not other elements in

the cloud infrastructure.

The Compute nodes is where the virtual machines are running. Since local traffic never leaves a Compute node, Snort will need to be installed here to capture it. Since there often are many Compute nodes in an OpenStack cloud, there is an element of load balancing by having Snort installed here. By listening on the interface br-int, all ranges of traffic is captured. This is the interface the virtual machines are connected to. The interface br-eth is a location where Snort is able to capture external and internal traffic, but not local.

Chapter 5

Evaluation framework

An evaluation framework has been created to evaluate different placements of Snort in OpenStack. The framework is score based where the solution with the lowest final score is considered the best choice among the alternatives. The framework has three dimensions: accuracy, complexity and potential performance influence. Accuracy describes what kind of traffic OpenStack is able to capture. These are the three kinds of traffic described earlier; external, internal and local. A higher accuracy is a positive trait, since it is able to detect attacks from more sources. A higher accuracy means a lower score given since lower scores are positive for the final result. Complexity is the number of Snort installations required in a solution. Complexity is a negative trait that will have a higher score value the more complex a solution is. Potential performance influence is how much a solution is expected to impact the performance of the cloud. The different levels of performance influence will be ranked in different categories. A higher influence will give a higher positive score, which will be negative for the final result.

5.1 Accuracy

Table 5.1 shows that a value will be added for each type of traffic Snort is able to capture: external, internal and local. Depending on how much accuracy is valued, A can be set to a value between -1 and -5. This value will then be added to the sum for each type of traffic the implementation is able to capture.

Table 5.1: Importance factor ranking

Features	Importance ranking
External	A
Internal	A
Local	A
Sum	

5.1.1 Other potential dimensions

For this analysis framework, the different types of traffic Snort is able to capture is included. But other aspects could be relevant to include in the analysis. The increase in CPU load and memory usage could be an important cost concern. The hardware needs to be dimensioned for the services running in OpenStack in addition to the extra resources used by Snort.

5.2 Complexity

The more installations of Snort that need to be configured and maintained, the more complex the setup will be. Installing and configuring two installations of Snort is about twice as much work as for one installation. But an assumption can be made that above a certain number of Snort installations, tools for automatic deployment will be developed. From there the increase in complexity will be minimal as the number of Snort installations increases. The calculation of complexity is shown in equation 5.1.

$$C_{complexity} = C_F - \frac{1}{S_{nort}} * C_F \tag{5.1}$$

To calculate the complexity value the complexity factor C_F is subtracted by 1 divided by the number of Snort installations the setup requires. This is then multiplied by the given complexity factor. The complexity factor is how much complexity will affect the final score. For some it may be burdensome to configure several or a large amount of Snort installations. Cost or time constraints can make a solution with high complexity less favourable. The complexity value can be set to a value between 1 and 5.

The relation between number of Snort installations and complexity is illustrated in Figure 5.1.

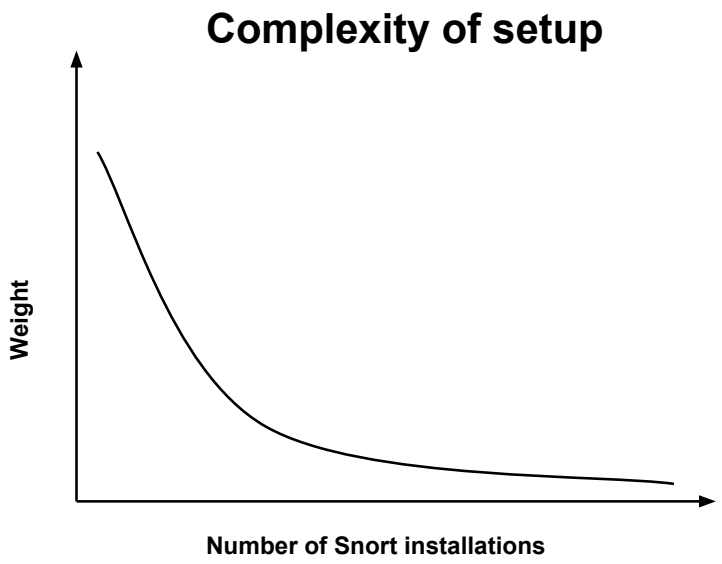

Figure 5.1: Diagram illustrating weight relationship with complexity

The fewest possible Snort installations, which is 1, will receive the lowest complexity value. After that the complexity value will increase as the number of Snort installations increases. Building on the assumption that after 3 or 4 Snort installations, automatic deployment tools will be developed, and the complexity value only increases slightly after that.

5.3 Potential performance influence

Different placements of Snort in the infrastructure will have different impact on performance. This is the impact on network performance and resources such as CPU utilization and memory usage. These performance influences are ranked in 4 categories: none, minor, medium, and major. None means that the flow of traffic is not interrupted and the packet inspection does not come at the cost of the resources used to power OpenStack. An example of this is a span port used on the physical switch to mirror network traffic to a separate machine with Snort installed. A ranking of minor means that elements in OpenStack is used to host Snort. These are elements there are one or few of, for example the Network node. This will only capture external traffic and is low in terms of resource usage. Medium performance influence is Snort placed at several elements in OpenStack. An example of this is the hypervisor on the Compute nodes. This can have a larger impact on performance since a wider range of traffic is captured and more installations of Snort is required. A major performance influence is a large number of Snort installations and a wide

range of captured traffic. This can be Snort located at each virtual machine, but can also be a combination of Snort installed on different elements in OpenStack. Since a higher performance influence is a negative trait, it is assigned positive values. The values increase exponentially the higher ranking a solution has on potential performance influence. This is shown in Table 5.2.

Table 5.2: Potential performance influence

Influence	Value	Check
None	1	X
Minor	2	X
Medium	4	X
Major	8	X

Chapter 6

Snort placements selected for evaluation

After exploring the networking of OpenStack in more detail, viable placements for Snort has been found. Four locations has been selected for evaluation using the evaluation framework.

6.1 Alternative 1: Snort at Network node

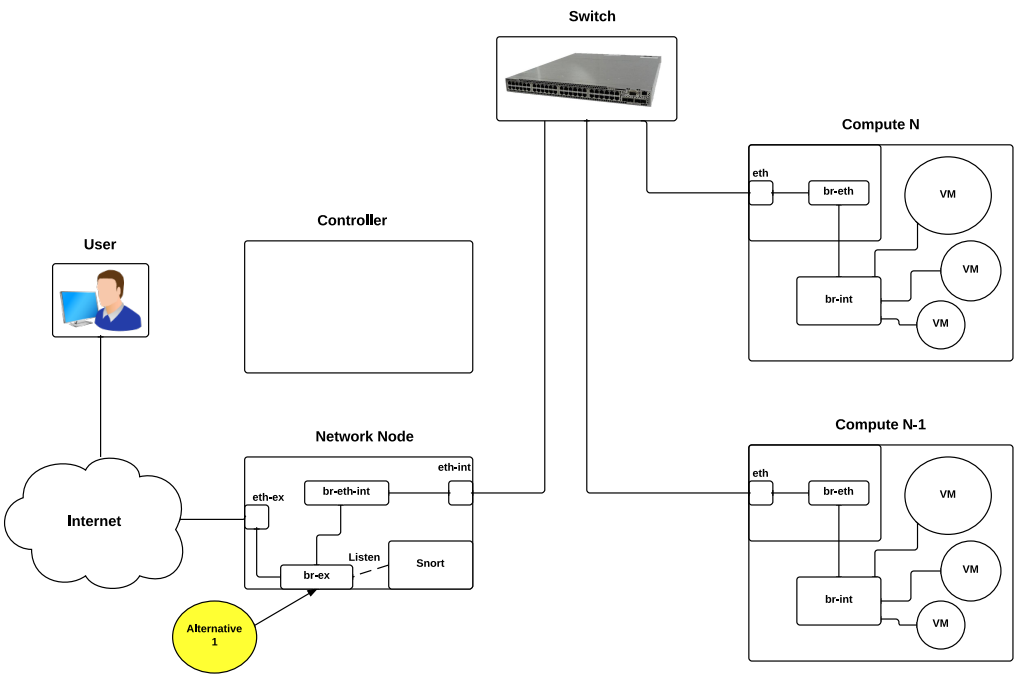

Figure 6.1: Alternative 1: Network node

As shown in Figure 6.1, Snort will here be placed at the network node. This will capture traffic coming from or are destined to the internet. The virtual bridge Br-Ex will be used to capture traffic.

6.2 Alternative 2: Snort box connected to switch

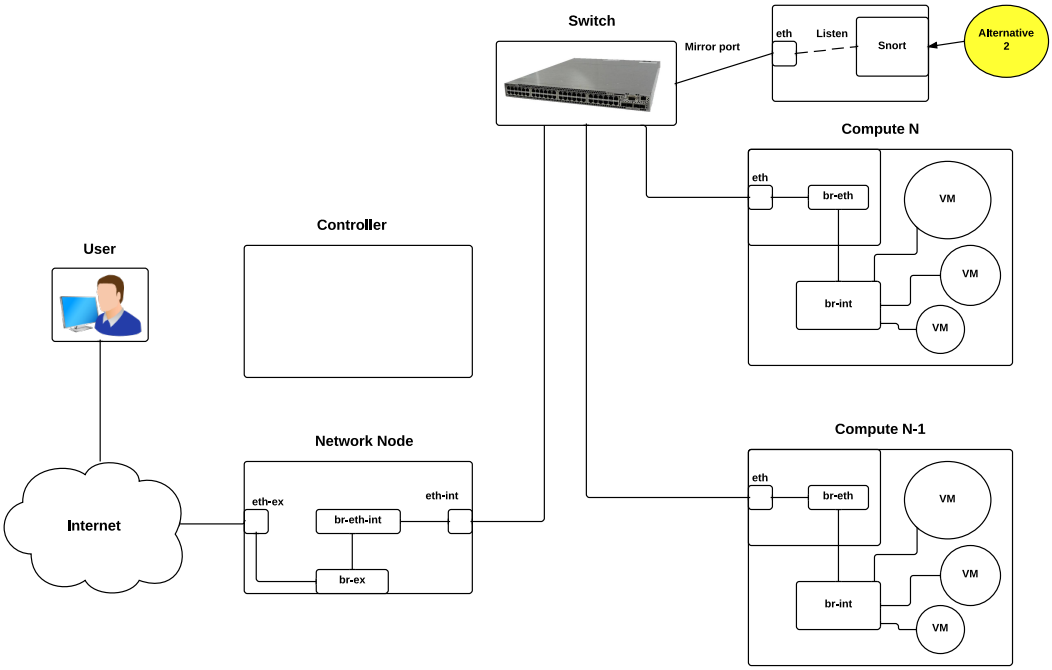

Figure 6.2: Alternative 2: Snort box connected to switch

Here Snort will be installed on a box and connected to the physical switch, illustrated in Figure 6.2. Using port mirroring network packets will be copied to the port the box is connected to. This will capture both external and internal traffic without affecting network performance. The box will have to use a 10Gbit Ethernet network card to match that of the switch.

6.3 Alternative 3: Snort connected to br-eth on Compute node

Snort will here be placed on on a Compute node listening to interface br-eth. This will enable Snort to listen for external and internal traffic. This comes at the expense of the processing power of the Compute node. This is shown in 6.3.

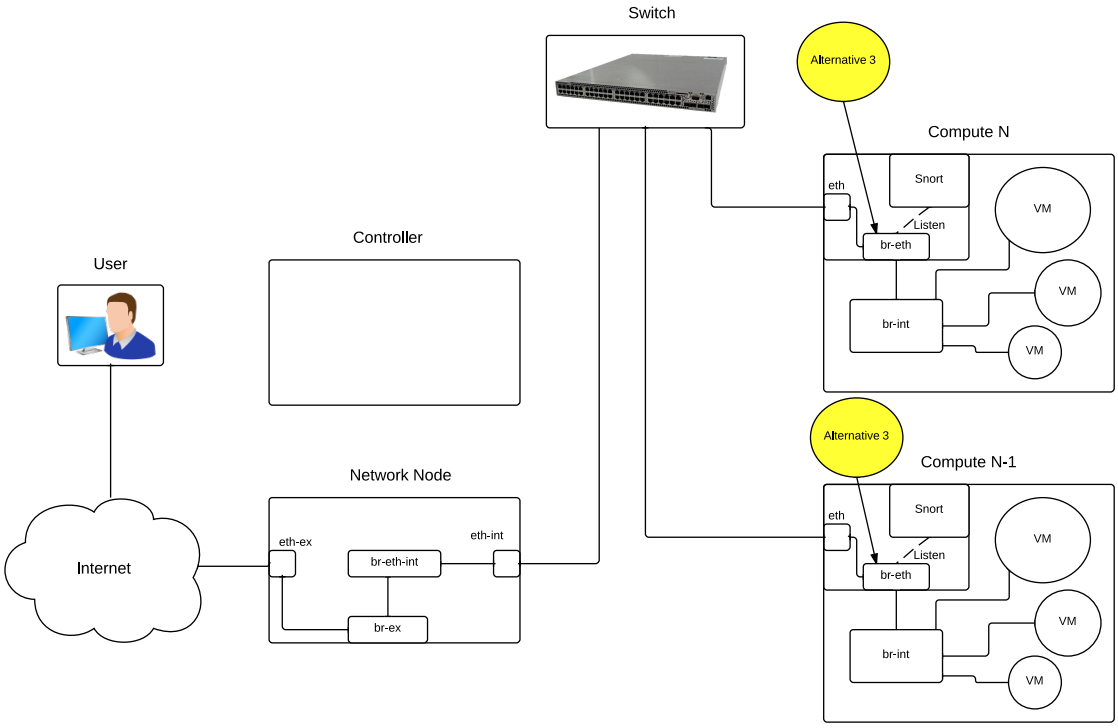

Figure 6.3: Alternative 3: Snort connected to br-eth on Compute node

6.4 Alternative 4: Snort connected to br-int at Compute node

The Snort placement is here on a Compute listening on interface br-int, as shown in Figure 6.4. In addition to external and internal traffic, Snort can now listen on local traffic, which is traffic between virtual machines on the same compute node.

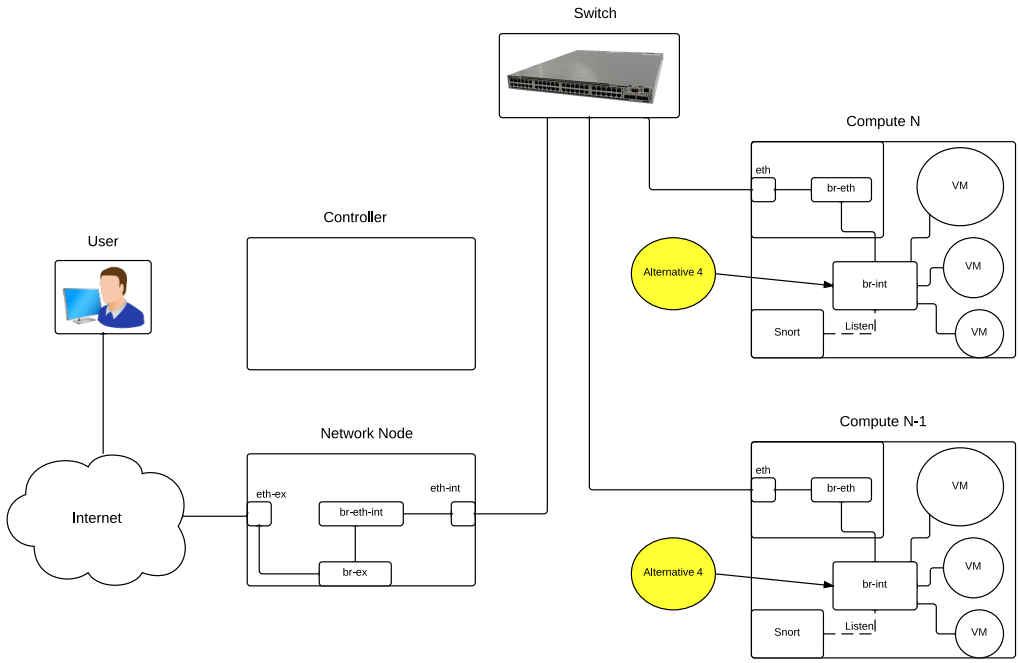

Figure 6.4: Alternative 3: Snort connected to br-int on Compute node

Chapter 7

Results and analysis of experiments

7.1 Experiment 1

Three tests was done in this experiment. The result is presented in Table 7.1. Snort was installed on the Network node listening on eth5, and Compute node 1 listening on eth7. The first test was to see if traffic coming from the Internet and to a virtual machine on Compute node 1 would trigger an alert on both Snort installations. A local rule was written in Snort to be triggered on the content "testing" in a TCP network packet. The written rule is listed below:

```
#-------------
# LOCAL RULES
#-------------

alert tcp any any -> any any (msg:"Testrule triggered";\
content:"testing"; priority:10; sid:99901;)
```

The setup for this experiment is illustrated in Figure 10.1. Two virtual machines were created on Compute node 1: test1 and test2. On test1 the webserver Apache 2 was installed with a text file named *testing.txt* placed in the */var/www/* folder. Using an external computer the IP address of test1 was entered into a web browser pointing to the text file. This generated a Snort alert on both Network1 and Compute node 1. In the second test a http request was sent from Compute node 2 to the web server on virtual machine test1, pointing to the same text file. This generated an alert on Snort at Compute node 1, but not on the Network node. This shows that internal traffic between Compute nodes is not sent to the Network node. In the third test a http request was sent from virtual machine test2 to test1, both residing on the same Compute node. This also generated an alert at Compute node 1, showing that local traffic is sent directly to other Compute nodes through the physical switch. This is presented in Table 7.1.

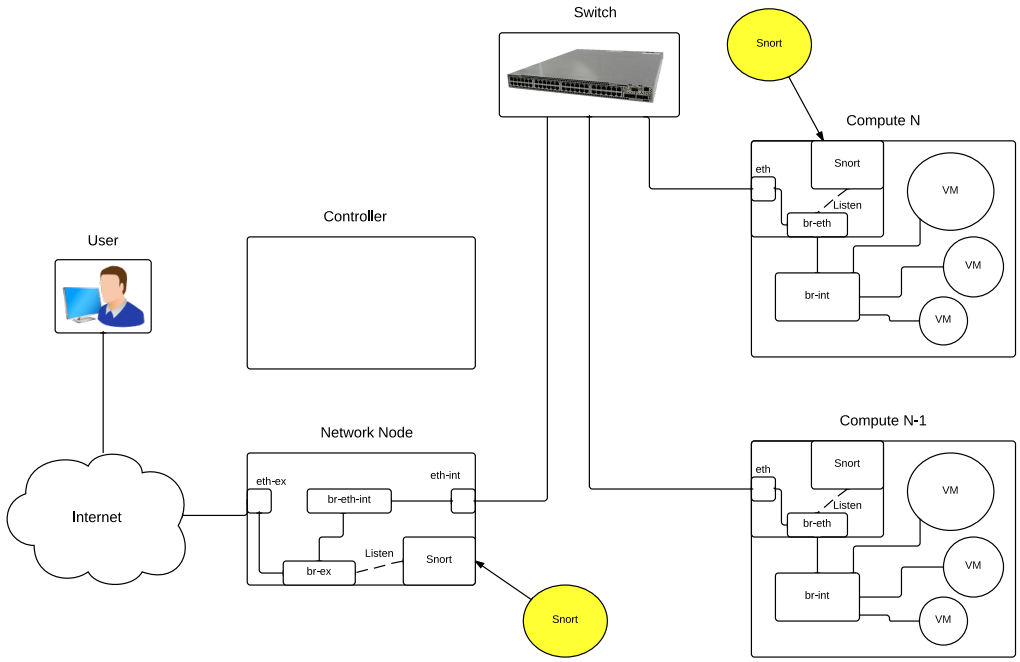

Figure 7.1: Diagram showing Snort placements in experiment 1

Table 7.1: Range of traffic captured by Snort

	Snort at Network node	Snort at hypervisor
External threat	X	X
Internal threat		X
Local threat		X

7.2 Experiment 2

To generate traffic, a script was set to download a 1.2GB ISO image of Linux Mint[15] from different sources. This ensured variety in rate of traffic sent through the network. The large file size also ensured enough traffic is received to have a noticeable impact on Snorts performance. The script used to download the ISO files is listed in Appendix A. The script used to log CPU and RAM usage used by the Snort process is listed in Appendix B. The script for logging traffic on an interface is listed in Appendix C. Below is a table of download sources selected from the Linux Mint download page for 64-bit Cinnamon edition.[14] The selected sources is presented with their location and IP address in Table 7.2. The setup for this experiment is illustrated in Figure 10.2.

Table 7.2: Sources used to download Linux Mint

Continent	Country	Name	IP address
Europe	Germany	Artfiles	80.252.110.38
Europe	Germany	FH Aachen	149.201.240.100
Europe	Germany	GWDG	2001:638:60f:110::1:1
Europe	Russia	Yandex Team	2a02:6b8::183
Europe	Sweden	DF - Lund University	194.47.250.18
Europe	France	Crifo.org	2001:bc8:30d4::900d:c0fe
Europe	France	Gwendal Le Bihan	195.154.174.66
Europe	United Kingdom	Bytemark Hosting	2001:41c8:20:5e6::10

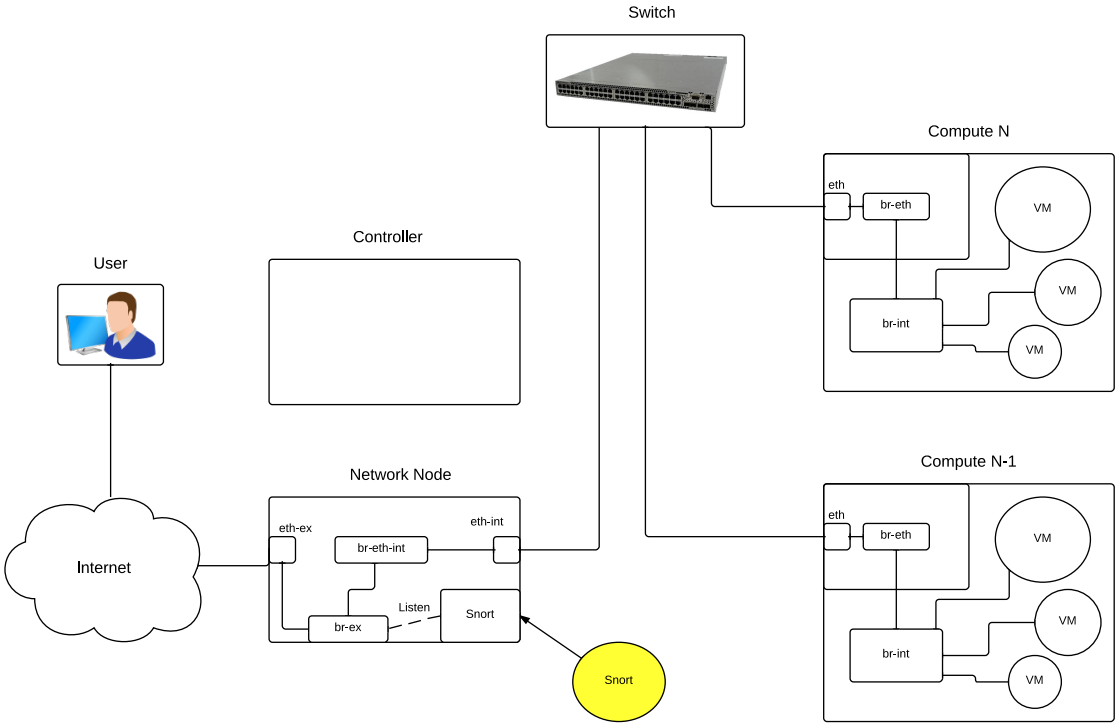

Figure 7.2: Diagram showing Snort placements in experiment 2

The script ran for approximately 80 minutes, looping through the list above twice. CPU and Memory usage used by the Snort process was recorded while it ran. A diagram showing the rate of traffic and CPU usage is shown in 7.3.

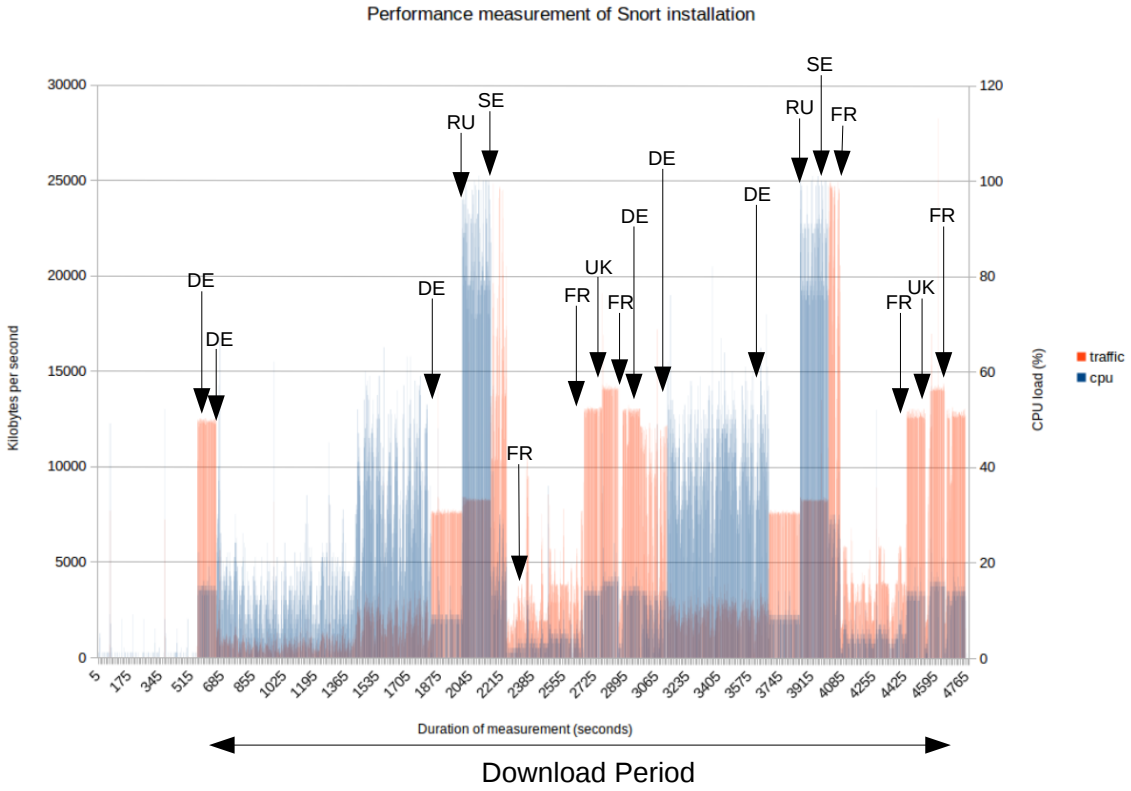

Figure 7.3: Result from performance measurement of Snort

Table 7.3: Performance measurements

	Value
Memory usage (pages)	153376
CPU usage pr KB transferred during download period	0.4673

The memory usage of the Snort process was constant throughout the experiment, staying at 153394 pages. The ratio of KB of traffic transferred and CPU usage is 0.4673. This is presented in Table 7.3.

As seen in Table 7.3 the ratio of CPU usage and traffic is drastically different, depending on the source it came from. This is a finding that have implications for the amount of resources a Snort installation is expected to use.

7.2.1 Repetition of experiment using two virtual machines

To see if the ratio of Kilobytes of traffic sent and CPU usage used by the Snort process changes with increased traffic two virtual machines were used. This will increase the overall traffic sent. The locations of Snort is shown in Figure 7.4.

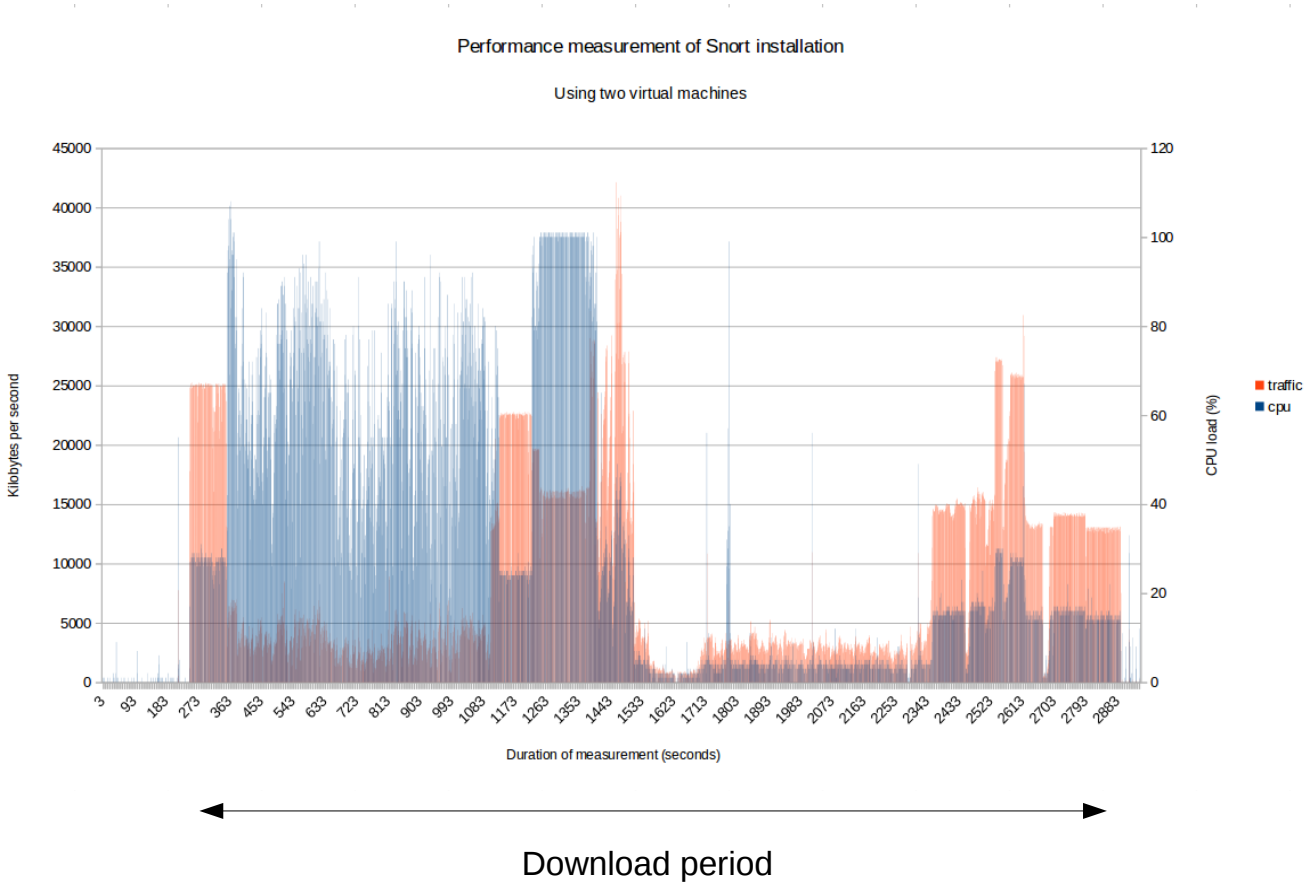

Figure 7.4: Result from performance measurement of Snort using two virtual machines

The memory used by the Snort process stays about the same, 153387 pages. The ratio of CPU usage and traffic is now 0.0796. This is presented in Table 7.4.

Table 7.4: Performance measurements

	Value
Memory usage (pages)	153387
CPU usage pr KB transferred during download period	0.0796

7.3 Follow-up experiment 1: Sources shown to generate high CPU load for Snort

In this experiment sources shown to have a high CPU load in relation to rate of traffic were selected to see if there is a significant difference in the resulting ratio from experiment 1. The chosen sources is shown in Table 8.2.

Table 7.5: Sources that generated high CPU load in experiment 1

Continent	Country	Name	IP address
Europe	Germany	FH Aachen	149.201.240.100
Europe	Russia	Yandex Team	2a02:6b8::183

As seen in Figure 7.5 the same pattern repeats where a high CPU load is seen relative to rate of KB transferred.

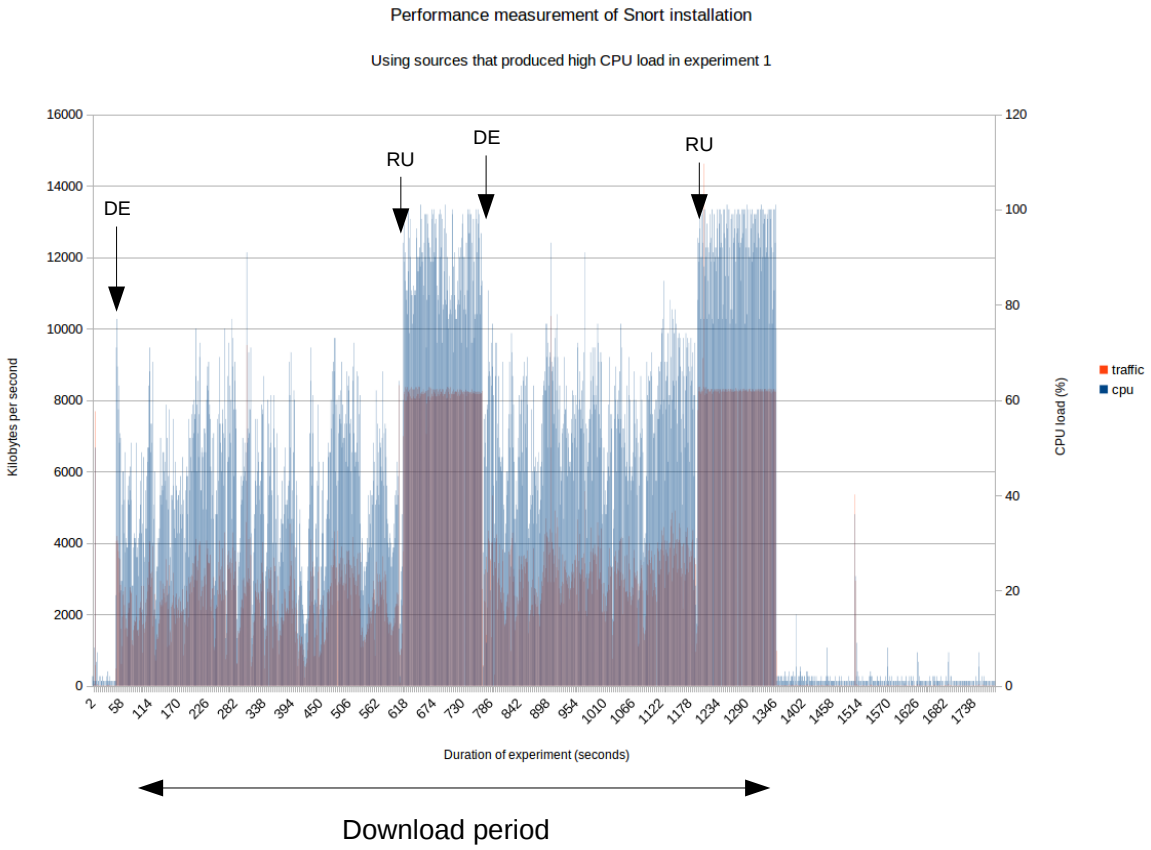

Figure 7.5: Result from performance measurement using sources that generated high CPU load in experiment 1

The ratio of CPU usage and traffic is now 2.3606, significantly higher than in experiment 1. The result is presented in Table 7.6.

Table 7.6: Performance measurements

	Value
Memory usage (pages)	153392
CPU usage pr KB transferred during download period	2.3606

7.4 Follow-up experiment 2: Sources shown to generate low CPU load for Snort

In this experiment sources shown to have a low CPU load in relation to rate of traffic were selected to see if there is a significant difference in the resulting ratio from experiment 1. The chosen sources is shown in Table 7.7.

Table 7.7: Sources that generated low CPU load in experiment 1

Continent	Country	Name	IP address
Europe	Germany	Artfiles	80.252.110.38
Europe	Germany	GWDG	2001:638:60f:110::1:1
Europe	Sweden	DF - Lund University	194.47.250.18
Europe	France	Crifo.org	2001:bc8:30d4::900d:c0fe
Europe	France	Gwendal Le Bihan	195.154.174.66

By picking sources that generated low CPU load in experiment 1, the same results is seen where there is a low CPU load relative to the traffic. This is shown in Figure 7.6.

The result from this experiment is presented in Table 7.8. The CPU usage pr KB transferred is now significantly lower than in follow-up experiment 1. The memory usage by the Snort process stay close to the same, at 153392 pages.

Table 7.8: Performance measurements

	Value
Memory usage (pages)	153392
CPU usage pr KB transferred	0.3619

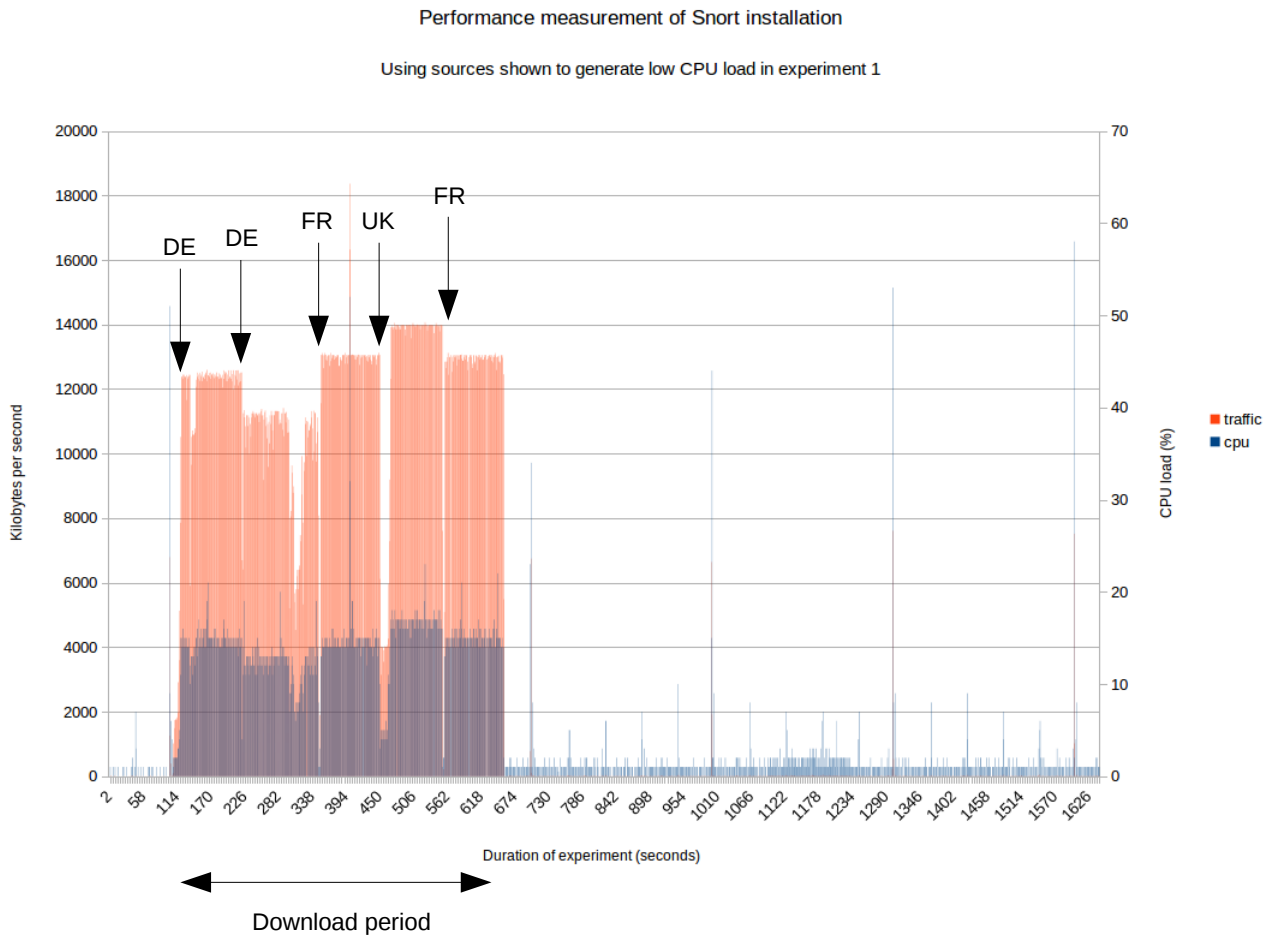

Figure 7.6: Result from performance measurement using sources that generated low CPU load in experiment 1

7.5 Summary

Experiment 1 showed the route each type of traffic takes through the infrastructure of OpenStack. This confirms the definitions set for the three types of traffic: external, internal, and local. With this the accuracy of a NID implementation can be determined when evaluating it with the evaluation framework.

Experiment 2 had an interesting finding. Some sources were shown to have a much higher CPU relative to the traffic they generated. This were confirmed in follow-up experiment 1 where sources shown to produce high CPU load followed the same trend. Sources with low CPU load also showed the same trend in follow-up experiment 2. Since the same file was downloaded from all sources, where traffic originated from has an impact

on the performance of Snort.

Chapter 8

Analysis of architectures using evaluation framework

The evaluation framework will be used to evaluate selected implementations of Snort. This will test the usefulness and highlight shortcomings of the evaluation framework. This evaluation will evolve around a scenario where a fictional company need to select a NID architecture for their OpenStack cloud. They have selected Snort as their network intrusion detection system and need to select where to implement it. The company has a high emphasis on security, so they want high accuracy for the architecture. This will translate to a low value for A an accuracy. They are willing to use time and resources to implement and manage a complex solution, but an overly complex one is not desirable. Their architecture consists of a Network node connected to four Compute nodes. After an analysis of the different architectures are completed, a summary will follow to show which implementation is best for this company. An evaluation of the evaluation framework itself will also be done to see if a revision is needed.

8.1 Alternative 1: Snort at Network node

As described in section 6.1, Snort is here placed at the Network node listening on the interface br-ex.

8.1.1 Accuracy

In this analysis the accuracy weight is set to -4, the second lowest value allowed by the framework. When placing Snort at the Network node in OpenStack, external traffic is captured, not Internal or Local.

8.1.2 Complexity

The complexity factor C_F will here be set to 2, meaning complexity is less of an issue when choosing a solution. The complexity is calculated below:

Table 8.1: Importance factor ranking for Alternative 1

Features	Importance ranking
External	-4
Internal	
Local	
Sum	-4

$$C_{complexity} = C_F - \frac{1}{S_{nort}} * C_F$$
$$C_{complexity} = 2 - \frac{1}{1} * 2$$
$$C_{complexity} = 0$$

8.1.3 Potential performance influence

Evaluating the potential performance influence is challenging in this scenario. The evaluation framework defines the section potential performance influence as the potential impact the architecture will have on network performance and resources such as CPU and RAM. Placing Snort at the Network node could potentially have a high impact on its performance since several Compute nodes are connected to it. The Network node serves as a gateway for external traffic between the Internet and the virtual machines running on Compute nodes. This makes it a choking point. But having Snort installed on the Network node means it will not have to analyse internal or local traffic. The potential performance influence is therefore set to medium.

Table 8.2: Potential performance influence

Influence	Value	Check
None	1	
Minor	2	
Medium	4	X
Major	8	

8.2 Alternative 2: Snort box connected to switch

As described in section 6.2, Snort is now installed on a separate machine. This machine is connected to the physical switch, and receives traffic coming from or are destined to the Compute nodes, through a mirror port.

8.2.1 Accuracy

When mirroring traffic passing through the switch to a box with Snort running on it, external and internal traffic is captured. The accuracy weight is set to -4.

Table 8.3: Importance factor ranking for Alternative 2

Features	Importance ranking
External	-4
Internal	-4
Local	
Sum	-8

8.2.2 Complexity

The complexity factor is set to 2. This lead to the following calculation:

$$C_{complexity} = C_F - \frac{1}{S_{nort}} * C_F$$
$$C_{complexity} = 2 - \frac{1}{1} * 2$$
$$C_{complexity} = 0$$

8.2.3 Potential performance influence

When having a separate box with Snort installed that receives mirrored traffic from the switch, the performance of the rest of the cloud is not affected. The box has its own resources, and since the traffic passing through the switch is only mirrored, there is no interruption of traffic flow. The potential performance influence is therefore none.

Table 8.4: Potential performance influence

Influence	Value	Check
None	1	X
Minor	2	
Medium	4	
Major	8	

8.3 Alternative 3: Snort connected to br-eth on Compute node

As described in section 6.2, Snort is installed on the Compute nodes listening on the interface br-eth.

8.3.1 Accuracy

Installing Snort on the hypervisor on a Compute node and make it listen to the interface br-eth enables it to listen on external and internal traffic. This give a score of 8 on accuracy.

Table 8.5: Importance factor ranking for Alternative 3

Features	Importance ranking
External	-4
Internal	-4
Local	
Sum	-8

8.3.2 Complexity

The complexity is set to 2. Since the company have four compute nodes, four installations of Snort is required. This is calculated below:

$$C_{omplexity} = C_F - \frac{1}{S_{nort}} * C_F$$
$$C_{omplexity} = 2 - \frac{1}{4} * 2$$
$$C_{omplexity} = 1.5$$

8.3.3 Potential performance influence

This architecture requires a Snort installation on each Compute node. This give a potential performance influence of medium.

Table 8.6: Potential performance influence

Influence	Value	Check
None	1	
Minor	2	
Medium	4	X
Major	8	

8.4 Alternative 4: Snort connected to br-int at Compute node

As described in section 6.4 Snort is installed on the Compute nodes listening on the interface br-int.

8.5 Accuracy

Installing Snort on the hypervisor on a Compute node, and make it listen on br-int, enables it to listen on external, internal, and local traffic. This is the full spectrum of traffic, and the architecture will therefore receive full score on accuracy.

Table 8.7: Importance factor ranking for Alternative 4

Features	Importance ranking
External	-4
Internal	-4
Local	-4
Sum	-12

8.5.1 Complexity

This architecture requires four installations of Snort. The complexity value is calculated below:

$$C_{complexity} = C_F - \frac{1}{S_{nort}} * C_F$$
$$C_{complexity} = 2 - \frac{1}{4} * 2$$
$$C_{complexity} = 1.5$$

8.5.2 Potential performance influence

This architecture requires a Snort installation on each Compute node. This give a potential performance influence of medium.

Table 8.8: Potential performance influence

Influence	Value	Check
None	1	
Minor	2	
Medium	4	X
Major	8	

8.6 Summary

The summary of this analysis is presented in Table 8.9. Alternative 4 achieves the lowest score on accuracy. Alternative 1 and 2 both get a score of 0 on complexity. This makes them the best choices if a solution with low complexity is wanted. Alternative 3 and 4 both get a score of 1.5 on. All alternatives received an equal score on potential performance influence. The alternative with the lowest sum is alternative 4. This is mainly because of a full score on accuracy, which was prioritized by the company with a low value for A.

Table 8.9: Summary table

	I	II	III	IV
Accuracy	-4	-8	-8	-12
Complexity	0	0	1.5	1.5
Potential performance influence	4	1	4	4
Sum	**0**	**-7**	**-2.5**	**-6.5**

A finding from this analysis is that accuracy have a greater impact on the final score than complexity. With a full weight on accuracy, which is -5, the largest score is -15. The score for complexity can only be close to 5 with a full weight and a large number of Snort installations. These values should be changed so they more closely matches each others ranges, and let the weight given to each or them be the differentiator. The potential performance influence should be changed to a weighted score. Different cloud providers will have different priorities when it comes to the performance of the cloud. Some may highly prioritize the performance and will not accept much interruption, while others place it lower than the other factors. The weight should enable the score to fall within a range close to or equal the other factors. This requires a revision of the evaluation framework.

Chapter 9

Evaluation framework, revision 2

In the analysis of architectures using the evaluation framework there were areas of improvements found. A weight should be added to potential performance influence so the users of the framework can choose how much they want to prioritize the performance of their cloud. The weights should adjusted so the three dimensions of the framework have an equal opportunity to influence the final score. It is the user of the framework that set the weight and determine what they want to prioritize.

9.1 Accuracy

Accuracy has a weight range between -1 and -5. This give a score range from -1 to -15. To even out the score ranges for the dimensions of the evaluation framework, accuracy will stay as it is, and the other weights will be change to match or closely match it.

Table 9.1: Importance factor ranking

Features	Importance ranking
External	A
Internal	A
Local	A

9.2 Complexity

The weight for complexity had a range between 1 and 5 in the first version of the evaluation framework. This give a score range between 0 and 5, depending on the weight and number of Snort installations. To closely match the score range of accuracy, the weight range will here be changed to 1 and 15. This will produce a score range from 0 to 15. The formula for calculating complexity will stay the same.

$$C_{complexity} = C_F - \frac{1}{S_{nort}} * C_F \tag{9.1}$$

9.3 Potential performance influence

The first version of the evaluation framework did not include a weight for potential performance influence. A weight range between 1 and 2 will be added here. The score is now calculated by multiplying the value for each level of performance influence with the performance factor P_F. This produces a new score range between 1 and 16, close to the other two dimensions.

Table 9.2: Potential performance influence

Influence	Value * P_F	Check
None	1 * P_F	X
Minor	2 * P_F	X
Medium	4 * P_F	X
Major	8 * P_F	X

Chapter 10

Analysis of Evaluation Framework, rev. 2

To test revision 2 of the evaluation framework, a new scenario has been selected. This is a company that need to choose between two alternatives for Snort placements in their OpenStack infrastructure. Their architecture is a Network node connected through a switch to eight Compute nodes. Their requirement for accuracy is high, fearing both external and internal threats to their services. They are willing to spend time and resources installing and maintaining a complex solution. Their services have a low tolerance for performance slowdowns, so a high weight for potential performance influence will be put here. This analysis will be used to check how useful the evaluation framework is for Snort placements not included in the first analysis. The first alternative is a combination of Alternative 1 and Alternative 3. This means that Snort will be installed on the Network node and on the hypervisor on each Compute node, listening on br-eth. This is illustrated in Figure 10.1.

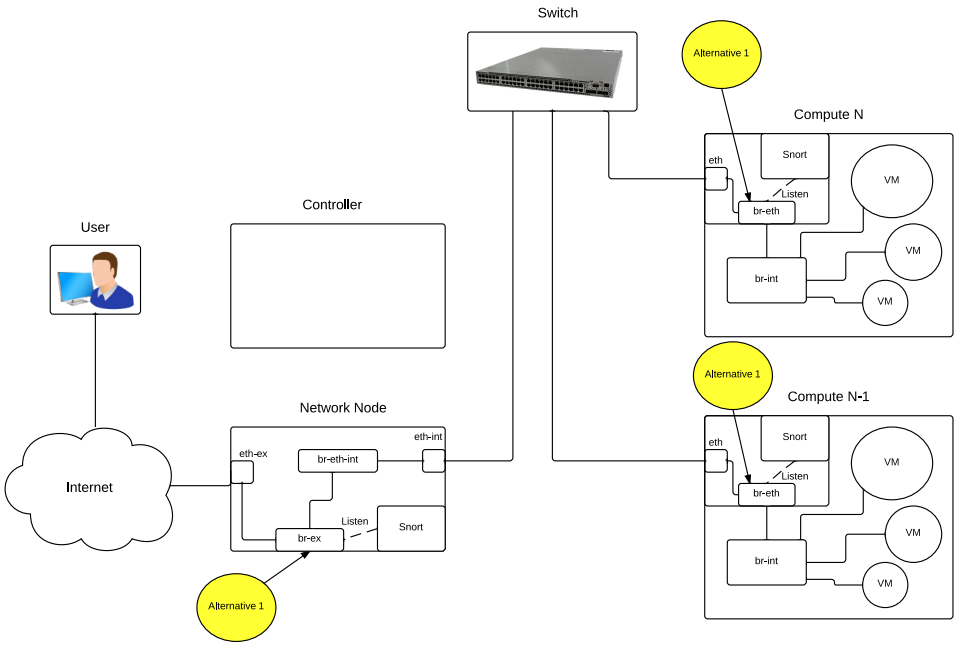

Figure 10.1: Diagram of the first alternative.

The second alternative is a combination of Alternative 2 and 4. This means that a box with Snort installed will be connected to a switch and network traffic will be mirrored to it through a span port. In addition to that, Snort will be installed on the hypervisor on each Compute node, listening on br-int. This alternative is illustrated in Figure 10.2.

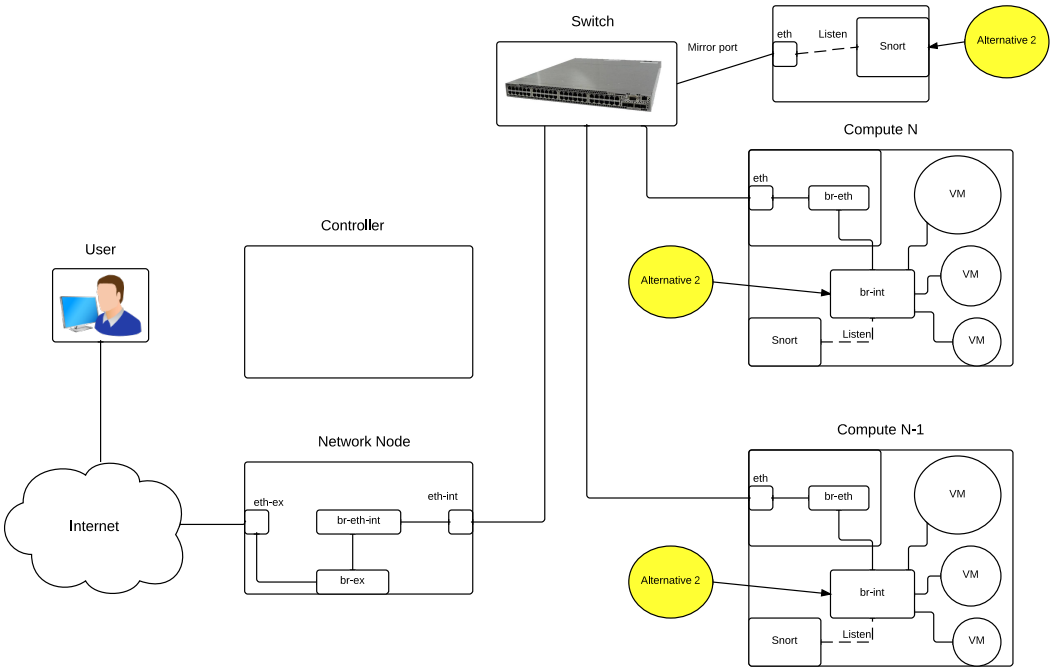

Figure 10.2: Diagram of the second alternative.

10.1 The first alternative: Combination of Alternative 1 and Alternative 3

10.1.1 Accuracy

With Snort located on the Network node and listening on br-eth on each Compute node, external and internal traffic is captured. The company's highest priority is accuracy, so a full weight of -5 will be added here. Added together, the score is -10.

Table 10.1: Importance factor ranking for the first alternative

Features	Importance ranking
External	-5
Internal	-5
Local	
Sum	-10

10.1.2 Complexity

As stated a complex solution is something the company is willing to spend time and resources on. Their priority is a solution with high accuracy and low performance influence. The weight for complexity is therefore set to 2. This produces a score of 1.778.

$$C_{complexity} = C_F - \frac{1}{S_{nort}} * C_F$$
$$C_{complexity} = 2 - \frac{1}{9} * 2$$
$$C_{complexity} = 1.778$$

10.1.3 Potential performance influence

The company prefers a small performance impact when selecting a Snort placement. The performance factor will therefore be 2. A combination of Alternative 1 and Alternative 2 constitutes a major performance influence.

Table 10.2: Potential performance influence

Influence	Value * P_F	Check
None		
Minor		
Medium		
Major	8 * 2	X

10.2 The second alternative: Combination of alternative 2 and 4

10.2.1 Accuracy

In this alternative a Snort box is connected to the switch, and Snort is installed on each Compute node listening on br-int. This captures all types of traffic: external, internal and local. The score for accuracy is therefore -15.

Table 10.3: Importance factor ranking for the first alternative

Features	Importance ranking
External	-5
Internal	-5
Local	-5
Sum	-15

10.2.2 Complexity

This alternative requires 9 Snort installations. 8 on each Compute node and 1 on the box connected to the switch. With a weight of 2 this give a score of 1.778.

$$C_{complexity} = C_F - \frac{1}{S_{nort}} * C_F$$
$$C_{complexity} = 2 - \frac{1}{9} * 2$$
$$C_{complexity} = 1.778$$

10.2.3 Potential performance influence

As seen in Table 10.4 this alternative have a medium performance influence. The box connected to the switch does not consume resources from the rest of the cloud, but each Compute node will have a Snort installation.

Table 10.4: Potential performance influence

Influence	Value * P_F	Check
None		
Minor		
Medium		4 * 2
Major		X

10.3 Summary

The final sum is presented in Table 10.5. The second alternative had the best score on accuracy, which was something the company wanted to prioritize. They both achieved the same score on accuracy, since they both required the same number of Snort installations. The first alternative had the highest potential performance influence. Using the evaluation framework, the second alternative is considered the best option for this company.

Table 10.5: Summary table

	First alternative	Second alternative
Accuracy	-10	-15
Complexity	1,778	1,778
Potential performance influence	16	8
Sum	**7.778**	**-5.222**

The framework does now include a weight for potential performance influence so the users of the framework can choose how much it influences the final score. The score ranges for each of the dimensions of the framework has also been adjusted to they closely matches each other.

Chapter 11

Discussion

An initial plan when starting this project was to investigate how different placements of network intrusion detection systems in cloud environments would affect detection capability and server performance. OpenStack was the cloud environment chosen for this project, and the network intrusion detection system was Snort. Should this have been the premise of the project, it could have turned into a benchmarking assignment for Snort with no clear direction. The numbers gathered from experiments would also mostly be specific to HiOAs Alto cloud. The rules for Snort is also periodically updated, which also have an effect on its performance.

The premise of the project was therefore changed to develop a framework to evaluate and rank different implementations of network intrusion detection systems in cloud environments. The evaluation framework is in this assignment used for OpenStack and Snort, but the method for evaluation is transferable to other cloud environments. The evaluation of different implementations of Snort in OpenStack can now be used to test the framework and find areas for improvement.

The problem statement asked how different implementations of network intrusion detection systems in cloud environments could be evaluated and ranked for specific use cases. To do this an evaluation framework was developed. To test the usefulness of this framework an analysis of different implementations of Snort in OpenStack were conducted. The online documentation available for OpenStacks networking were first expected to be enough to find good places to implement Snort, but were soon found to be insufficient.

This lead to an experiment, Experiment 1, whose purpose was to map the flow of different types of traffic in OpenStack. A manual mapping of OpenStacks network interfaces at various locations were also done where documentation fell short. When the mapping of OpenStacks networking were completed, viable placements for Snort could be decided. Experiment 2 and follow-up experiments 1 and 2 showed that the source traffic originates from has an effect on Snorts performance. Given the large

difference seen in CPU utilization this could have been included in the evaluation framework. This may have been in the form of differentiating the types of clouds by how much external traffic they are expected to receive.

11.1 Extension to the evaluation framework: 4 types of cloud environments

The more external traffic traffic a cloud receives, the higher the changes is that it receives traffic from sources that generate high CPU load by Snort. The cloud environments can then be divided into 4 categories with that in mind.

The first type is a private cloud with focus on secrecy. Examples of this is a bank or research institution. Content transferred to this cloud is usually limited and from limited sources. The second type is a private cloud with less focus on secrecy. An example of this is an IT company using a private cloud for their internal projects. There is some restrictions on content transferred to this cloud, but less than that of a bank. The third type is a private cloud used by different departments within a company. These departments are mostly free to decide how to use their share of the clouds resources. The fourth type is a public cloud. This could be the well known Amazon EC2 or Google Compute engine. The resources of these clouds are rented out to customers who use it for their own purposes. These clouds are expected to receive large amounts of traffic from many different sources.

This can be incorporated in the rest of the evaluation framework by adding a privacy value. An example of this is shown in Table 11.1. A higher privacy value will mean that the NID implementation can be expected to have a higher impact on the performance of the cloud. Either by influencing the value for potential performance influence, or to be included as a standalone value.

Table 11.1: Privacy table

Type of cloud	**Privacy**
Type 1	0
Type 2	1
Type 3	2
Type 4	4

11.2 Discussion about future work

An early idea about this project was to offer Network Intrusion Detection as a Service (NIDaaS) in OpenStack. Users of the cloud could then have an option in the web interface Horizon if they want to enable NID for their virtual machines. This would certainly be a very interesting feature to introduce in OpenStack, but also has its problems. The assignment would be a very technical one involving reading the source code for OpenStack in order to find ways to implement it. A large portion of the time would be spend coding and testing the code. It also has some questions related to it about the direction of this approach. The users of cloud environments can often be considered less knowledgeable about security than the systems administrators running it. Important services can therefore go unprotected without the systems administrators knowledge.

This approach would also yield one way to protect a cloud environment, and could be difficult to incorporate with other security measures. The direction chosen for this project was therefore to evaluate different implementations from a cloud provider perspective, not a user perspective. The strength and weaknesses of each implementations can with this method be evaluated and compared. NIDaaS can therefore be a future project by someone wanting to implement this. The information gathered in this project about the details of OpenStacks networking will prove useful information, and the evaluation framework can be used to evaluate different alternatives.

11.3 Discussion about related work

The evaluation framework is useful for cloud providers running OpenStack and need to decide on a network intrusion detection solution. The NIDS used in this project is Snort, so the evaluation framework is developed with that in mind. The evaluation framework should be general enough to be used in other cloud environments, even using other network intrusion detection systems. Some modifications to the framework may then have to be done.

No such framework was found when searching for related literature, so this is something new this project has produced. Research into security and cloud environments is ongoing. Related work found for Snort were seen to focus on improving Snort modules. Modi et al[16] showed that high performance for Snort can be achieved while using a combination of signatures and anomaly detection. Other experimental techniques, such as work done by Xing et al[24], target the protection of cloud network from another angle. By reconfiguring the network, new countermeasures against attacks is introduced. This is interesting work for finding new ways to improve the security of cloud environments. The work done in this project aims to be of use to those wanting to use Snort to protect their

OpenStack cloud and need to choose between different architectures. It is also a basis for further work into evaluation of NID implementations in cloud environments.

Chapter 12

Conclusion

Cloud computing has seen a tremendous growth for several years now, and this growth is expected to continue. How to best protect services hosted in clouds is a hot topic of research, and also an issue more and more companies are facing. This work has demonstrated an approach for evaluating and ranking different alternatives for network intrusion detection implementations in cloud environments. Using weighed score the users of the framework can influence the outcome by selecting their priorities. Different companies will have different priorities for what they expect of their solution, so the framework reflects that.

By mapping the networking of OpenStack in more detail than provided by the online documentation, several good alternatives for where to place network intrusion detection systems were found. These alternatives, along with combinations of them, were evaluated using the evaluation framework. This shows the feasibility of using the framework in different usage scenarios.

The problem statement asked "How can different implementations of network intrusion detection systems in cloud environments be evaluated and ranked for specific use cases?". The evaluation framework produced in this thesis shows that this is possible using weighted scores for the different aspects of the implementations. The evaluation framework is open for further improvements if new aspects of a solution need to be incorporated.

Chapter 13

Appendices

13.1 Appendix A: Script do download ISO files

getiso.pl

```
#!/usr/bin/perl

# our needed packages
use strict "vars";
use Getopt::Std;

# Global variables
my $VERBOSE = 0;
my $DEBUG = 0;

# command line options

my $opt_string = "vdhf:";
getopts( "$opt_string", \my %opt ) or usage() and exit(1);

$VERBOSE = 1 if $opt{'v'};
$DEBUG = 1 if $opt{'d'};
if ( $opt{'h'} ){
    usage();
    exit 0;
}


####################################
# Main part

my $count = 0;
my $time = time;
my $logfile = "$time.log.txt";

while( $count < 2 ){

```

```
33
34
35      open(FILE,"mirrors2.txt") or die("Unable to open file: $1")\
36      ;
37
38      while( my $line = <FILE> ){
39
40          chomp($line);
41          $time = time;
42
43          open(LOG,">>$logfile");
44          print LOG "$time;$line\n";
45          close(LOG);
46
47          system("wget -O /dev/null $line");
48
49          $count++;
50
51      }
52  }
53
54
55
56  debug("script is finished, exiting...\n");
57  exit 0;
58
```

13.2 Appendix B: Snort CPU and RAM usage

stats_snort.pl

```
#!/usr/bin/perl

# our needed packages
use strict "vars";
use Getopt::Std;
use POSIX ();

# Global variables
my $VERBOSE = 0;
my $DEBUG = 0;

# command line options

my $opt_string = "vdhf:";
getopts( "$opt_string", \my %opt ) or usage() and exit(1);

$VERBOSE = 1 if $opt{'v'};
$DEBUG = 1 if $opt{'d'};
if ( $opt{'h'} ){
    usage();
    exit 0;
}

# variables
my $PID = pid();
die("Snort is not running") unless $PID;
verbose("PID for Snort is $PID\n");

my $MEM;
my $time1 = time;
my $time2;
my $cpu1;
my $cpu2;
my $count;

####################################
# Main part
my $logfile = "$time1.snort.log.txt";

my $clock_ticks = POSIX::sysconf( &POSIX::_SC_CLK_TCK );
verbose("clock tics: $clock_ticks\n");


while( 1 ){

```

```
46      if ( $count ){
47
48          # printing cpu load and memory usage to logfile
49          $time2 = time;
50          $cpu2 = cpuusage();
51          my $cpuDiff = $cpu2 - $cpu1;
52          my $diffTime = $time2 - $time1;
53          my $memusage = memusage();
54          verbose("Current memory usage: $memusage\n");
55          my $pcpu =  (($cpuDiff * 100) / $clock_ticks) / \
56          $diffTime) unless $diffTime == 0;
57          my $pcpu2 = sprintf("%.3f",$pcpu);
58          verbose("CPU load $pcpu2\n");
59
60          open(LOG,">>$logfile");
61          print LOG "$time2:$pcpu2:$memusage\n";
62          close(LOG);
63
64          $time1 = $time2;
65          $cpu1 = $cpu2;
66          sleep 1;
67
68      }else{
69          $time1 = time;
70          $count = 1;
71          $cpu1 = cpuusage();
72          sleep 1;
73
74          system("rm logfile.txt");
75          open(LOG,">>logfile.txt");
76          print LOG "Timestamp:Cpu load:Memory usage\n";
77          close(LOG);
78      }
79  }
80
81
82
83  debug("script is finished, exiting...\n");
84  exit 0;
85
86  ################################
87  # subroutines
88
89  sub cpuusage {
90
91      open(STAT,"/proc/$PID/stat") or die("Unable to open file:\
92      $!");
93
```

```
94      my @stat = split(/\s+/,<STAT>);
95
96      my $total = $stat[13] + $stat[14];
97      return $total;
98  }
99
100 sub memusage {
101
102     # finding memory usage of perlbal by using PID
103     open(FILE,"/proc/$PID/statm") or die("Unable to open file:\
104     $!");
105
106     while( my $line = <FILE> ){
107
108         my @elements = split(/\s+/,$line);
109
110         return $elements[0];
111         last;
112     }
113 }
114
115 sub pid {
116
117     # finding PID
118     open(FILE,"ps aux |");
119     while( my $line = <FILE> ){
120
121         if( $line =~ /^\S+\s+(\S+)\s+.*eth5/ ){
122             verbose("Successful if test. PID is $1\n");
123             return $1;
124             last;
125         }
126     }
127     close(FILE);
128 }
129
```

13.3 Appendix C: Traffic logger

traffic.pl

```
#!/usr/bin/perl

# our needed packages
use strict "vars";
use Getopt::Std;

# Global variables
my $VERBOSE = 0;
my $DEBUG = 0;

# command line options

my $opt_string = "vdhf:";
getopts( "$opt_string", \my %opt ) or usage() and exit(1);

$VERBOSE = 1 if $opt{'v'};
$DEBUG = 1 if $opt{'d'};
if ( $opt{'h'} ){
    usage();
    exit 0;
}



####################################
# Main part

my $time = time;

my $logfile = "$time.traffc.log.txt";

while( 1 ){

    open(FILE,"ifconfig eth5 |") or die("Unable to open file:\
    $!");
    while(my $line = <FILE>){

        if( $line =~ /RX bytes:(\d+).+TX bytes:(\d+).+/ ){

            $time = time;
            open(LOG,">>$logfile");
            print LOG "$time:$1:$2\n";
            verbose("Line: $1 $2\n");
            close LOG;
            last;
```

```
        }
    }
    close FILE;
    sleep 1;
}


debug("script is finished, exiting...\n");
exit 0;

```

Bibliography

[1] Amazon. *Amazon EC2 homepage*. 2013. URL: http://aws.amazon.com/ec2/.

[2] Inc Amazon Web Services. *Amazon DevPay*. 2014. URL: http://aws.amazon.com/devpay/.

[3] Inc Amazon Web Services. *Amazon Web Services*. 2014. URL: http://aws.amazon.com/.

[4] Inc Amazon Web Services. *AWS Marketplace webpage*. 2014. URL: https://aws.amazon.com/marketplace.

[5] inc Citrix Systems. *XenServer hompage*. 2014. URL: http://www.xenserver.org/.

[6] CloudPassage. *CloudPassage Halo*. 2014. URL: https://aws.amazon.com/marketplace/pp/B008KTGRKK/ref=srh_res_product_title?ie=UTF8&sr=0-4&qid=1396344217945.

[7] Louis Columbus. *Gartner Predicts Infrastructure Services Will Accelerate Cloud Computing Growth*. 2013. URL: http://www.forbes.com/sites/louiscolumbus/2013/02/19/gartner-predicts-infrastructure-services-will-accelerate-cloud-computing-growth/.

[8] Quinstreet Enterprise. *The 7 Layers of the OSI Model*. 2014. URL: http://www.webopedia.com/quick_ref/OSI_Layers.asp.

[9] Inc Eucalyptus Systems. *Eucalyptus homepage*. 2014. URL: https://www.eucalyptus.com/.

[10] Google. *Google Compute Engine homepage*. 2013. URL: https://cloud.google.com/products/compute-engine/.

[11] MetaFlows inc. *MetaFlows Security System for EC2*. 2014. URL: https://aws.amazon.com/marketplace/pp/B008MAO9SE/ref=srh_res_product_title?ie=UTF8&sr=0-3&qid=1395675542931.

[12] Zhe Li, Weiqing Sun and Lingfeng Wang. 'A neural network based distributed intrusion detection system on cloud platform'. In: *Cloud Computing and Intelligent Systems (CCIS), 2012 IEEE 2nd International Conference on*. Vol. 01. Oct. 2012, pp. 75–79. DOI: 10.1109/CCIS.2012.6664371.

[13] Alert Logic. *Alert Logic Threat Manager for AWS (EC2)*. 2014. URL: https://aws.amazon.com/marketplace/pp/B00DZYG3D6/ref=srh_res_product_title?ie=UTF8&sr=0-2&qid=1395321442246.

[14] Linux Mint. *Linux mint download page*. 2014. URL: http://www.linuxmint.com/edition.php?id=144.

[15] Linux Mint. *Linux Mint hompepage*. 2014. URL: http://www.linuxmint.com/.

[16] C.N. Modi et al. 'Bayesian Classifier and Snort based network intrusion detection system in cloud computing'. In: *Computing Communication Networking Technologies (ICCCNT), 2012 Third International Conference on*. 2012, pp. 1–7. DOI: 10.1109/ICCCNT.2012.6396086.

[17] Oslo and Akershus University College of Applied Sciences. *HiOA homepage*. 2014. URL: http://www.hioa.no/eng/.

[18] OpenNebula Project. *OpenNebula homepage*. 2014. URL: http://opennebula.org/.

[19] The OpenStack project. *OpenStack homepage*. 2014. URL: http://www.openstack.org/.

[20] Sony Shetty. *Gartner Says Cloud Computing Will Become the Bulk of New IT Spend by 2016*. 2013. URL: http://www.gartner.com/newsroom/id/2613015.

[21] Sourcefire. *Sourcefire homepage*. 2013. URL: http://www.sourcefire.com/.

[22] inc. Sourcefire. *Snort homepage*. 2014. URL: http://www.snort.org/.

[23] Tcpdump/Libpcap. *Tcpdump homepage*. 2014. URL: http://www.tcpdump.org/.

[24] Tianyi Xing et al. 'SnortFlow: A OpenFlow-Based Intrusion Prevention System in Cloud Environment'. In: *Research and Educational Experiment Workshop (GREE), 2013 Second GENI*. 2013, pp. 89–92. DOI: 10.1109/GREE.2013.25.